```
#@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.
```

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

```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
```

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

```
2.3.0
```

Saved successfully!

```
    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


#clears any previous variables - to prevent other models' interference
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)

#create the properly structured dataset
train_set = windowed_dataset(x_train, window_size, batch_size=128, shuffle_buffer=shuffle_buf

model = tf.keras.models.Sequential([
  #lambda layers allow us to add some arbritary code
  #rnn's require us to enter a 3 dimensional input
  #our windowed data is only 2 dimensions, so using this we can add a third dimension
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]), #cuz any size must be accepted
  #a simple RNN - need to specify return_sequences to true cuz the output needs to be
```

Saved successfully!

```
  #a simple RNN - need to specify return sequences to true cuz the output needs to be
  #fed into the next RNN
  tf.keras.layers.SimpleRNN(40, return_sequences=True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1),
  #activation function of lambdas a by default tanh, which gives a value between -1 and 1
  #so in order to bring it to scale (values in window are 10, 20, 30s...) we multiply by hund
  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)
#huber loss is a specialised loss function that isn't so sensitive to outliers
#since time series data tends to become really noisy - this loss function is preferred
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```


Saved successfully!

```
Epoch 1/100
8/8 [==============================] - 0s 18ms/step - loss: 195.5726 - mae: 196.0726
Epoch 2/100
8/8 [==============================] - 0s 11ms/step - loss: 194.7820 - mae: 195.2820
Epoch 3/100
8/8 [==============================] - 0s 12ms/step - loss: 193.5109 - mae: 194.0109
Epoch 4/100
8/8 [==============================] - 0s 10ms/step - loss: 191.9081 - mae: 192.4081
Epoch 5/100
8/8 [==============================] - 0s 11ms/step - loss: 190.0235 - mae: 190.5235
Epoch 6/100
8/8 [==============================] - 0s 11ms/step - loss: 187.8582 - mae: 188.3582
Epoch 7/100
8/8 [==============================] - 0s 11ms/step - loss: 185.3787 - mae: 185.8787
Epoch 8/100
8/8 [==============================] - 0s 10ms/step - loss: 182.5483 - mae: 183.0483
Epoch 9/100
8/8 [==============================] - 0s 10ms/step - loss: 179.3126 - mae: 179.8126
Epoch 10/100
8/8 [==============================] - 0s 10ms/step - loss: 175.6017 - mae: 176.1017
Epoch 11/100
8/8 [==============================] - 0s 11ms/step - loss: 171.3243 - mae: 171.8243
Epoch 12/100
8/8 [==============================] - 0s 11ms/step - loss: 166.3521 - mae: 166.8521
Epoch 13/100
8/8 [==============================] - 0s 11ms/step - loss: 160.5517 - mae: 161.0517
Epoch 14/100
8/8 [==============================] - 0s 12ms/step - loss: 153.6378 - mae: 154.1378
Epoch 15/100
8/8 [==============================] - 0s 10ms/step - loss: 145.3261 - mae: 145.8261
Epoch 16/100
8/8 [==============================] - 0s 11ms/step - loss: 135.0351 - mae: 135.5351
Epoch 17/100
8/8 [==============================] - 0s 10ms/step - loss: 121.9920 - mae: 122.4920
Epoch 18/100
                                    ====] - 0s 11ms/step - loss: 105.3426 - mae: 105.8426
Saved successfully!
                                    ====] - 0s 11ms/step - loss: 84.1254 - mae: 84.6254
Epoch 20/100
8/8 [==============================] - 0s 11ms/step - loss: 57.2710 - mae: 57.7710
Epoch 21/100
8/8 [==============================] - 0s 12ms/step - loss: 32.0323 - mae: 32.5289
Epoch 22/100
8/8 [==============================] - 0s 11ms/step - loss: 24.3013 - mae: 24.7973
Epoch 23/100
8/8 [==============================] - 0s 11ms/step - loss: 23.3490 - mae: 23.8461
Epoch 24/100
8/8 [==============================] - 0s 10ms/step - loss: 21.1135 - mae: 21.6101
Epoch 25/100
8/8 [==============================] - 0s 10ms/step - loss: 17.1272 - mae: 17.6220
Epoch 26/100
8/8 [==============================] - 0s 11ms/step - loss: 13.8473 - mae: 14.3365
Epoch 27/100
8/8 [==============================] - 0s 11ms/step - loss: 10.7749 - mae: 11.2631
Epoch 28/100
8/8 [==============================] - 0s 11ms/step - loss: 8.7659 - mae: 9.2524
Epoch 29/100
```

```
8/8 [==============================] - 0s 11ms/step - loss: 7.9783 - mae: 8.4596
Epoch 30/100
8/8 [==============================] - 0s 11ms/step - loss: 7.8824 - mae: 8.3651
Epoch 31/100
8/8 [==============================] - 0s 11ms/step - loss: 7.8424 - mae: 8.3231
Epoch 32/100
8/8 [==============================] - 0s 12ms/step - loss: 7.7293 - mae: 8.2061
Epoch 33/100
8/8 [==============================] - 0s 11ms/step - loss: 7.6896 - mae: 8.1682
Epoch 34/100
8/8 [==============================] - 0s 11ms/step - loss: 7.5853 - mae: 8.0646
Epoch 35/100
8/8 [==============================] - 0s 10ms/step - loss: 7.5172 - mae: 7.9977
Epoch 36/100
8/8 [==============================] - 0s 11ms/step - loss: 7.4070 - mae: 7.8849
Epoch 37/100
8/8 [==============================] - 0s 11ms/step - loss: 7.3480 - mae: 7.8265
Epoch 38/100
8/8 [==============================] - 0s 10ms/step - loss: 7.2938 - mae: 7.7685
Epoch 39/100
8/8 [==============================] - 0s 12ms/step - loss: 7.2491 - mae: 7.7239
Epoch 40/100
8/8 [==============================] - 0s 11ms/step - loss: 7.2260 - mae: 7.7028
Epoch 41/100
8/8 [==============================] - 0s 12ms/step - loss: 7.2844 - mae: 7.7629
Epoch 42/100
8/8 [==============================] - 0s 11ms/step - loss: 7.4529 - mae: 7.9378
Epoch 43/100
8/8 [==============================] - 0s 11ms/step - loss: 7.1585 - mae: 7.6390
Epoch 44/100
8/8 [==============================] - 0s 11ms/step - loss: 7.1436 - mae: 7.6195
Epoch 45/100
8/8 [==============================] - 0s 11ms/step - loss: 7.0569 - mae: 7.5397
Epoch 46/100
8/8 [==============================] - 0s 11ms/step - loss: 6.9329 - mae: 7.4142
Epoch 47/100
                                    ====] - 0s 10ms/step - loss: 6.7453 - mae: 7.2217
                                    ====] - 0s 11ms/step - loss: 7.4523 - mae: 7.9375
Epoch 49/100
8/8 [==============================] - 0s 11ms/step - loss: 7.2814 - mae: 7.7655
Epoch 50/100
8/8 [==============================] - 0s 10ms/step - loss: 6.9146 - mae: 7.3947
Epoch 51/100
8/8 [==============================] - 0s 10ms/step - loss: 7.0549 - mae: 7.5381
Epoch 52/100
8/8 [==============================] - 0s 11ms/step - loss: 7.4218 - mae: 7.9044
Epoch 53/100
8/8 [==============================] - 0s 10ms/step - loss: 6.9011 - mae: 7.3845
Epoch 54/100
8/8 [==============================] - 0s 11ms/step - loss: 6.7600 - mae: 7.2387
Epoch 55/100
8/8 [==============================] - 0s 10ms/step - loss: 6.6565 - mae: 7.1352
Epoch 56/100
8/8 [==============================] - 0s 10ms/step - loss: 6.1894 - mae: 6.6657
Epoch 57/100
8/8 [==============================] - 0s 10ms/step - loss: 6.8613 - mae: 7.3413
Epoch 58/100
```

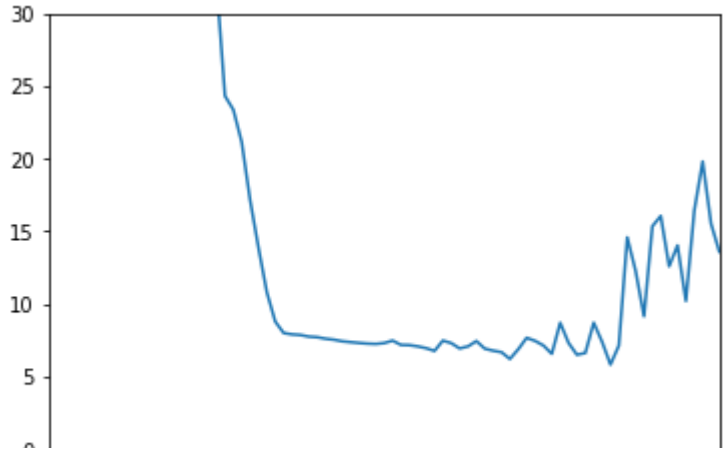
Saved successfully!

```
8/8 [==============================] - 0s 11ms/step - loss: 7.6439 - mae: 8.1289
Epoch 59/100
8/8 [==============================] - 0s 12ms/step - loss: 7.4355 - mae: 7.9232
Epoch 60/100
8/8 [==============================] - 0s 11ms/step - loss: 7.1180 - mae: 7.6021
Epoch 61/100
8/8 [==============================] - 0s 11ms/step - loss: 6.5446 - mae: 7.0291
Epoch 62/100
8/8 [==============================] - 0s 11ms/step - loss: 8.6818 - mae: 9.1692
Epoch 63/100
8/8 [==============================] - 0s 12ms/step - loss: 7.2914 - mae: 7.7792
Epoch 64/100
8/8 [==============================] - 0s 12ms/step - loss: 6.4782 - mae: 6.9644
Epoch 65/100
8/8 [==============================] - 0s 10ms/step - loss: 6.6052 - mae: 7.0900
Epoch 66/100
8/8 [==============================] - 0s 11ms/step - loss: 8.6873 - mae: 9.1772
Epoch 67/100
8/8 [==============================] - 0s 10ms/step - loss: 7.3687 - mae: 7.8522
Epoch 68/100
8/8 [==============================] - 0s 10ms/step - loss: 5.8037 - mae: 6.2825
Epoch 69/100
8/8 [==============================] - 0s 10ms/step - loss: 7.1136 - mae: 7.5967
Epoch 70/100
8/8 [==============================] - 0s 11ms/step - loss: 14.5646 - mae: 15.0591
Epoch 71/100
8/8 [==============================] - 0s 12ms/step - loss: 12.2519 - mae: 12.7449
Epoch 72/100
8/8 [==============================] - 0s 10ms/step - loss: 9.1278 - mae: 9.6167
Epoch 73/100
8/8 [==============================] - 0s 11ms/step - loss: 15.3273 - mae: 15.8214
Epoch 74/100
8/8 [==============================] - 0s 11ms/step - loss: 16.0371 - mae: 16.5327
Epoch 75/100
8/8 [==============================] - 0s 10ms/step - loss: 12.5683 - mae: 13.0615
Epoch 76/100
8/8 [==============================] - 0s 10ms/step - loss: 14.0105 - mae: 14.5047
Epoch 77/100
8/8 [==============================] - 0s 11ms/step - loss: 10.1825 - mae: 10.6726
Epoch 78/100
8/8 [==============================] - 0s 11ms/step - loss: 16.4089 - mae: 16.9054
Epoch 79/100
8/8 [==============================] - 0s 10ms/step - loss: 19.7855 - mae: 20.2826
Epoch 80/100
8/8 [==============================] - 0s 11ms/step - loss: 15.5033 - mae: 15.9982
Epoch 81/100
8/8 [==============================] - 0s 11ms/step - loss: 13.5496 - mae: 14.0409
Epoch 82/100
8/8 [==============================] - 0s 11ms/step - loss: 14.2568 - mae: 14.7495
Epoch 83/100
```

Saved successfully!

```
plt.semilogx(history.history["lr"], history.history["loss"])
#plot loss graph to obtain the optimal value for the learning rate(done by the callback)
plt.axis([1e-8, 1e-4, 0, 30])
```

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




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

dataset = windowed_dataset(x_train, window_size, batch_size=128, shuffle_buffer=shuffle_buffe

model = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]),
  tf.keras.layers.SimpleRNN(40, return_sequences=True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])

optimizer = tf.keras.optimizers.SGD(lr=5e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
```

Saved successfully!

```
00)
```

```
Epoch 1/400
8/8 [==============================] - 0s 12ms/step - loss: 81.7535 - mae: 82.2535
Epoch 2/400
8/8 [==============================] - 0s 10ms/step - loss: 21.9358 - mae: 22.4301
Epoch 3/400
8/8 [==============================] - 0s 10ms/step - loss: 15.9200 - mae: 16.4143
Epoch 4/400
8/8 [==============================] - 0s 11ms/step - loss: 12.2879 - mae: 12.7814
Epoch 5/400
8/8 [==============================] - 0s 10ms/step - loss: 10.1503 - mae: 10.6405
Epoch 6/400
8/8 [==============================] - 0s 12ms/step - loss: 16.3163 - mae: 16.8110
Epoch 7/400
8/8 [==============================] - 0s 11ms/step - loss: 13.7287 - mae: 14.2229
Epoch 8/400
8/8 [==============================] - 0s 11ms/step - loss: 9.6225 - mae: 10.1128
Epoch 9/400
8/8 [==============================] - 0s 10ms/step - loss: 8.9765 - mae: 9.4663
Epoch 10/400
8/8 [==============================] - 0s 10ms/step - loss: 7.9001 - mae: 8.3863
Epoch 11/400
8/8 [==============================] - 0s 11ms/step - loss: 9.1519 - mae: 9.6434
Epoch 12/400
8/8 [==============================] - 0s 10ms/step - loss: 9.3649 - mae: 9.8516
Epoch 13/400
8/8 [==============================] - 0s 11ms/step - loss: 7.9262 - mae: 8.4150
Epoch 14/400
8/8 [==============================] - 0s 10ms/step - loss: 6.9853 - mae: 7.4712
Epoch 15/400
8/8 [==============================] - 0s 11ms/step - loss: 7.2190 - mae: 7.7054
Epoch 16/400
8/8 [==============================] - 0s 10ms/step - loss: 7.5239 - mae: 8.0119
Epoch 17/400
8/8 [==============================] - 0s 10ms/step - loss: 6.2945 - mae: 6.7777
Epoch 18/400
                                    ==] - 0s 11ms/step - loss: 7.7614 - mae: 8.2495
                                    ==] - 0s 11ms/step - loss: 6.6822 - mae: 7.1692
Epoch 20/400
8/8 [==============================] - 0s 10ms/step - loss: 6.2704 - mae: 6.7545
Epoch 21/400
8/8 [==============================] - 0s 14ms/step - loss: 6.0768 - mae: 6.5560
Epoch 22/400
8/8 [==============================] - 0s 11ms/step - loss: 7.3321 - mae: 7.8209
Epoch 23/400
8/8 [==============================] - 0s 10ms/step - loss: 6.8335 - mae: 7.3189
Epoch 24/400
8/8 [==============================] - 0s 11ms/step - loss: 6.2629 - mae: 6.7502
Epoch 25/400
8/8 [==============================] - 0s 11ms/step - loss: 7.7601 - mae: 8.2474
Epoch 26/400
8/8 [==============================] - 0s 10ms/step - loss: 14.3939 - mae: 14.8888
Epoch 27/400
8/8 [==============================] - 0s 11ms/step - loss: 9.9131 - mae: 10.4011
Epoch 28/400
8/8 [==============================] - 0s 11ms/step - loss: 10.2379 - mae: 10.7282
Epoch 29/400
```

Saved successfully!