```
# LSTM (One to Many Multiple Numeric Feature)
# ===========================================

# https://stackabuse.com/solving-sequence-problems-with-lstm-in-keras-part-2/

%tensorflow_version 2.x

import tensorflow as tf
tf.__version__
```

```
TensorFlow 2.x selected.
'2.0.0'
```

```
# univariate lstm example
import tensorflow as tf
import numpy as np
from numpy import array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Bidirectional, Flatten
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.python.keras.callbacks import TensorBoard
# from tensorflow.keras.regularizers import l2

import matplotlib.pyplot as plt
from time import time
```

```
# define dataset
X1 = list()
X2 = list()
X = list()
Y = list()

X1 = [(x+1)*2 for x in range(25)]
X2 = [(x+1)*3 for x in range(25)]

for x1, x2 in zip(X1, X2):
    output_vector = list()
    output_vector.append(x1+1)
    output_vector.append(x2+1)
    Y.append(output_vector)

X = np.column_stack((X1, X2))
print(X)
```

```
[[ 2  3]
 [ 4  6]
 [ 6  9]
 [ 8 12]
 [10 15]
 [12 18]
 [14 21]
 [16 24]
 [18 27]
 [20 30]
 [22 33]
 [24 36]
 [26 39]
 [28 42]
 [30 45]
 [32 48]
 [34 51]
 [36 54]
 [38 57]
 [40 60]
 [42 63]
 [44 66]
 [46 69]
 [48 72]
 [50 75]]
```

```
1 X = np.array(X)
2 y = np.array(Y)
3
4 X = X.astype('float32')
5 y = y.astype('float32')
```

```
1 X[:3], y[:3]
```

```
(array([[2., 3.],
        [4., 6.],
        [6., 9.]], dtype=float32), array([[ 3.,  4.],
        [ 5.,  7.],
        [ 7., 10.]], dtype=float32))
```

```
1
2 print("X.shape : {}".format(X.shape))
3
4 # reshape from [samples, timesteps] into [samples, timesteps, features]
5 X = X.reshape((X.shape[0], 1, 2))
6
7 print("X.shape2 : {}".format(X.shape))
8
```

```
X.shape : (25, 2)
X.shape2 : (25, 1, 2)
```

```
# X = tf.cast(X,tf.float32)
# y = tf.cast(y,tf.float32)
```

```
# %load_ext tensorboard
# tensorboard = TensorBoard(log_dir="logs/{}".format(time()), histogram_freq=1)
# %tensorboard --logdir logs
```

```
# es = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=5, verbose=1, mode
```

```
# define model

model = Sequential()
model.add(Bidirectional(LSTM(50, activation='relu', input_shape=(1, 2), return_sequ
model.add(Dense(2))
model.compile(optimizer='adam', loss='mse', metrics=['mse'])
# history = model.fit(X, y, epochs=200, validation_split=0.2, batch_size=8, verbose
history = model.fit(X, y, epochs=1000, validation_split=0.2, batch_size=3, verbose=

model.summary()
```

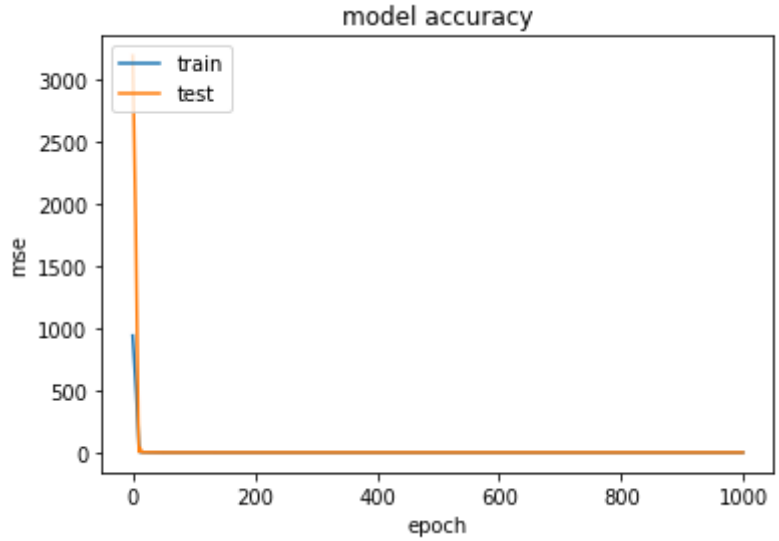
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
bidirectional_1 (Bidirection multiple                  21200     
_________________________________________________________________
dense_1 (Dense)              multiple                  202       
=================================================================
Total params: 21,402
Trainable params: 21,402
Non-trainable params: 0
_________________________________________________________________
```

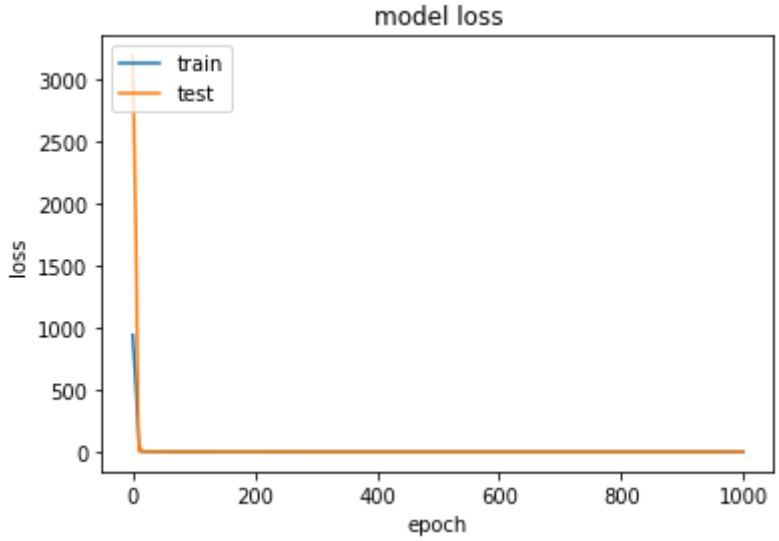
```
# fit model
# model.fit(X, y, epochs=500, validation_split=0.2, verbose=1, callbacks=[tensorboa
# history = model.fit(X, y, epochs=500, validation_split=0.2, verbose=0, callbacks=
```

```
# list all data in history
print(history.history.keys())

# summarize history for accuracy
plt.plot(history.history['mse'])
plt.plot(history.history['val_mse'])
plt.title('model accuracy')
plt.ylabel('mse')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'mse', 'val_loss', 'val_mse'])
```






```
# demonstrate prediction
x_input = array([40, 60])
print("x_input.shape {}".format(x_input.shape))

x_input = x_input.reshape((1, 1, 2))
print("x_input.shape2 {}".format(x_input.shape))

x_input = tf.cast(x_input,tf.float32)

print("expected : 41, 61")

yhat = model.predict(x_input, verbose=0)
print("yhat : ", yhat)
```

```
print( 'yhat :  ', yhat)
```

```
x_input.shape (2,)
x_input.shape2 (1, 1, 2)
expected : 41, 61
yhat :  [[41.00509  61.020744]]
```

```
1
```