Name : Nidhi Singh

Reg. No. : 22MAI0015

DEEP LEARNING LAB

DIGITAL ASSIGNMENT - V

COURCE CODE : MCSE603P

LAB_9_1: https://colab.research.google.com/drive/1lZfMsHrhQCHTsOop1BO-dsk5r6vu-EG-?usp=sharing

LAB_9_2: https://colab.research.google.com/drive/1U6_MbI0Ei8-vHA38Cza5EntCM0hjLY3t?usp=drive_link

Timeseries Forecasting for Weather Prediction

```
!nvidia-smi
```

```
Tue May 16 06:55:11 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   38C    P8     9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
```

```
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Step 1: Initializing

```
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import numpy as np
tf.keras.backend.clear_session()
tf.random.set_seed(123)
np.random.seed(123)
```

Step 2: Import the dataset

```
from zipfile import ZipFile
import os


uri = "https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip"
zip_path = keras.utils.get_file(origin=uri, fname="jena_climate_2009_2016.csv.zip")
zip_file = ZipFile(zip_path)
zip_file.extractall()
csv_path = "jena_climate_2009_2016.csv"
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip
13568290/13568290 [==============================] - 2s 0us/step
```

```
df = pd.read_csv(csv_path)
```

```
titles = [
"Pressure",
"Temperature",
"Temperature in Kelvin",
"Temperature (dew point)",
"Relative Humidity",
"Saturation vapor pressure",
```

```
"Vapor pressure",
"Vapor pressure deficit",
"Specific humidity",
"Water vapor concentration",
"Airtight",
"Wind speed",
"Maximum wind speed",
"Wind direction in degrees",
]


feature_keys = [
"p (mbar)",
"T (degC)",
"Tpot (K)",
"Tdew (degC)",
"rh (%)",
"VPmax (mbar)",
"VPact (mbar)",
"VPdef (mbar)",
"sh (g/kg)",
"H2OC (mmol/mol)",
"rho (g/m**3)",
"wv (m/s)",
"max. wv (m/s)",
"wd (deg)",
]


colors = [
"blue",
"orange",
"green",
"red",
"purple",
"brown",
"pink",
"gray",
"olive",
"cyan",
]


date_time_key = "Date Time"
```

```
def show_raw_visualization(data):
    time_data = data[date_time_key]
    fig, axes = plt.subplots(
    nrows=7, ncols=2, figsize=(15, 20), dpi=80, facecolor="w", edgecolor="k"
    )
    for i in range(len(feature_keys)):
        key = feature_keys[i]
        c = colors[i % (len(colors))]
        t_data = data[key]
        t_data.index = time_data
        t_data.head()
        ax = t_data.plot(
        ax=axes[i // 2, i % 2],
        color=c,
        title="{} - {}".format(titles[i], key),
        rot=25,
        )
        ax.legend([titles[i]])
    plt.tight_layout()
```

Step 3: Data Preprocessing

```
split_fraction = 0.715
train_split = int(split_fraction * int(df.shape[0]))
step = 6


past = 720
future = 72
learning_rate = 0.001
batch_size = 256
epochs = 10


def normalize(data, train_split):
    data_mean = data[:train_split].mean(axis=0)
    data_std = data[:train_split].std(axis=0)
    return (data - data_mean) / data_std
```

Step 4: Feature Engineering using Heat-Map

```
def show_heatmap(data):
   plt.matshow(data.corr())
   plt.xticks(range(data.shape[1]), data.columns, fontsize=14, rotation=90)
   plt.gca().xaxis.tick_bottom()
   plt.yticks(range(data.shape[1]), data.columns, fontsize=14)
   cb = plt.colorbar()
   cb.ax.tick_params(labelsize=14)
   plt.title("Feature Correlation Heatmap", fontsize=14)
   plt.show()
show_heatmap(df)
```

```
<ipython-input-14-699d41daef6b>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a futur
  plt.matshow(data.corr())
```


```
print(
"The selected parameters are:",
", ".join([titles[i] for i in [0, 1, 5, 7, 8, 10, 11]]),
)
```

```
The selected parameters are: Pressure, Temperature, Saturation vapor pressure, Vapor pressure deficit, Specific humidity, Airtight, Wind speed
```


```
selected_features = [feature_keys[i] for i in [0, 1, 5, 7, 8, 10, 11]]
features = df[selected_features]
features.index = df[date_time_key]
features.head()
```

Date Time	p (mbar)	T (degC)	VPmax (mbar)	VPdef (mbar)	sh (g/kg)	rho (g/m**3)	wv (m/s)
01.01.2009 00:10:00	996.52	-8.02	3.33	0.22	1.94	1307.75	1.03
01.01.2009 00:20:00	996.57	-8.41	3.23	0.21	1.89	1309.80	0.72
01.01.2009 00:30:00	996.53	-8.51	3.21	0.20	1.88	1310.24	0.19
01.01.2009 00:40:00	996.51	-8.31	3.26	0.19	1.92	1309.19	0.34
01.01.2009 00:50:00	996.51	-8.27	3.27	0.19	1.92	1309.00	0.32

```
features = normalize(features.values, train_split)
features = pd.DataFrame(features)
features.head()
```

	0	1	2	3	4	5	6
0	0.955451	-2.000020	-1.319782	-0.788479	-1.500927	2.237658	-0.732997
1	0.961528	-2.045185	-1.332806	-0.790561	-1.519521	2.287838	-0.936002
2	0.956666	-2.056766	-1.335410	-0.792642	-1.523239	2.298608	-1.283076
3	0.954236	-2.033604	-1.328898	-0.794724	-1.508364	2.272906	-1.184847
4	0.954236	-2.028972	-1.327596	-0.794724	-1.508364	2.268256	-1.197944

```
train_data = features.loc[0 : train_split - 1]
val_data = features.loc[train_split:]
```

Task: List the discarded features and justify the reason

```
#Pressure, Temperature, Saturation vapor pressure, Vapor pressure deficit, Specific humidity, Airtight, Wind speed

print(
"The discarded parameters are:",
", ".join([titles[i] for i in [2, 3, 4, 6, 9,10, 11,12,13]]),
)
```

```
The discarded parameters are: Temperature in Kelvin, Temperature (dew point), Relative Humidity, Vapor pressure, Water vapor concentration, Airtight, Winc
```

Step 5: Splitting Training and Validation Data

```
start = past + future
end = start + train_split


x_train = train_data[[i for i in range(7)]].values
y_train = features.iloc[start:end][[1]]
sequence_length = int(past / step)


dataset_train = keras.preprocessing.timeseries_dataset_from_array(
x_train,
y_train,
sequence_length=sequence_length,
sampling_rate=step,
batch_size=batch_size,
)


x_end = len(val_data) - past - future
label_start = train_split + past + future
x_val = val_data.iloc[:x_end][[i for i in range(7)]].values
y_val = features.iloc[label_start:][[1]]
```

```
dataset_val = keras.preprocessing.timeseries_dataset_from_array(
x_val,
y_val,
sequence_length=sequence_length,
sampling_rate=step,
batch_size=batch_size,
)


for batch in dataset_train.take(1):
    inputs, targets = batch


print("Input shape:", inputs.numpy().shape)
print("Target shape:", targets.numpy().shape)
```

```
Input shape: (256, 120, 7)
Target shape: (256, 1)
```

Step 6: Designing LSTM Architecture

```
inputs = keras.layers.Input(shape=(inputs.shape[1], inputs.shape[2]))
lstm_out = keras.layers.LSTM(32)(inputs)
cnn_out = keras.layers.Dense(16)(lstm_out)
outputs = keras.layers.Dense(1)(cnn_out)


model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer=keras.optimizers.Adam(learning_rate=learning_rate), loss="mse")
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 120, 7)]          0

 lstm (LSTM)                 (None, 32)                5120

 dense (Dense)               (None, 16)                528

 dense_1 (Dense)             (None, 1)                 17

=================================================================
```

```
Total params: 5,665
Trainable params: 5,665
Non-trainable params: 0
_________________________________________________________________
```

Step 7: Training the model

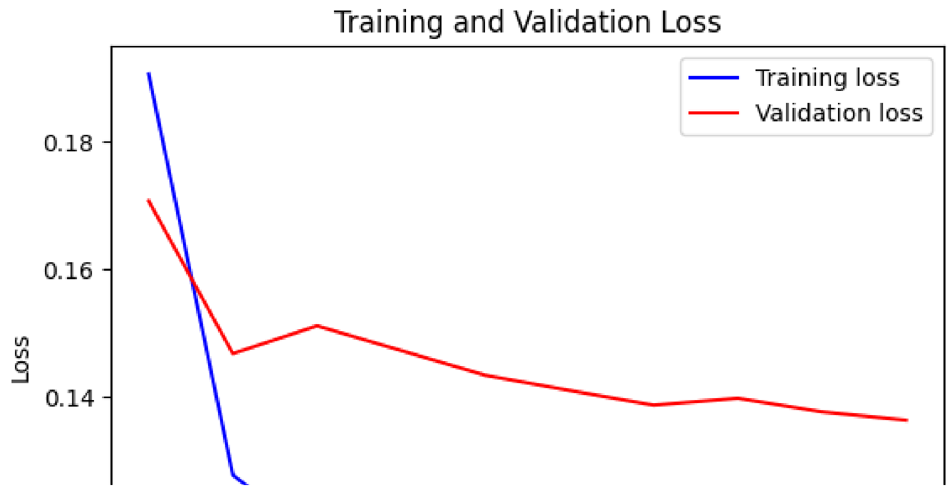
```
path_checkpoint = "model_checkpoint.h5"
es_callback = keras.callbacks.EarlyStopping(monitor="val_loss", min_delta=0, patience=5)
modelckpt_callback = keras.callbacks.ModelCheckpoint(
monitor="val_loss",
filepath=path_checkpoint,
verbose=1,
save_weights_only=True,
save_best_only=True,
)
history = model.fit(
dataset_train,
epochs=epochs,
validation_data=dataset_val,
callbacks=[es_callback, modelckpt_callback],
)
```

```
Epoch 1/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1906
Epoch 1: val_loss improved from inf to 0.17075, saving model to model_checkpoint.h5
1172/1172 [==============================] - 79s 61ms/step - loss: 0.1906 - val_loss: 0.1708
Epoch 2/10
1165/1172 [============================>.] - ETA: 0s - loss: 0.1283
Epoch 2: val_loss improved from 0.17075 to 0.14683, saving model to model_checkpoint.h5
1172/1172 [==============================] - 75s 64ms/step - loss: 0.1278 - val_loss: 0.1468
Epoch 3/10
1168/1172 [============================>.] - ETA: 0s - loss: 0.1178
Epoch 3: val_loss did not improve from 0.14683
1172/1172 [==============================] - 74s 63ms/step - loss: 0.1176 - val_loss: 0.1512
Epoch 4/10
1172/1172 [==============================] - ETA: 0s - loss: 0.1142
Epoch 4: val_loss did not improve from 0.14683
1172/1172 [==============================] - 77s 66ms/step - loss: 0.1142 - val_loss: 0.1473
Epoch 5/10
1171/1172 [============================>.] - ETA: 0s - loss: 0.1112
Epoch 5: val_loss improved from 0.14683 to 0.14344, saving model to model_checkpoint.h5
1172/1172 [==============================] - 75s 64ms/step - loss: 0.1112 - val_loss: 0.1434
Epoch 6/10
```

```
1167/1172 [============================>.] - ETA: 0s - loss: 0.1092
Epoch 6: val_loss improved from 0.14344 to 0.14106, saving model to model_checkpoint.h5
1172/1172 [==============================] - 76s 65ms/step - loss: 0.1090 - val_loss: 0.1411
Epoch 7/10
1169/1172 [============================>.] - ETA: 0s - loss: 0.1076
Epoch 7: val_loss improved from 0.14106 to 0.13876, saving model to model_checkpoint.h5
1172/1172 [==============================] - 87s 74ms/step - loss: 0.1074 - val_loss: 0.1388
Epoch 8/10
1171/1172 [============================>.] - ETA: 0s - loss: 0.1054
Epoch 8: val_loss did not improve from 0.13876
1172/1172 [==============================] - 73s 62ms/step - loss: 0.1054 - val_loss: 0.1398
Epoch 9/10
1170/1172 [============================>.] - ETA: 0s - loss: 0.1040
Epoch 9: val_loss improved from 0.13876 to 0.13770, saving model to model_checkpoint.h5
1172/1172 [==============================] - 76s 64ms/step - loss: 0.1040 - val_loss: 0.1377
Epoch 10/10
1171/1172 [============================>.] - ETA: 0s - loss: 0.1023
Epoch 10: val_loss improved from 0.13770 to 0.13640, saving model to model_checkpoint.h5
1172/1172 [==============================] - 68s 58ms/step - loss: 0.1023 - val_loss: 0.1364
```

```
def visualize_loss(history, title):
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, "b", label="Training loss")
    plt.plot(epochs, val_loss, "r", label="Validation loss")
    plt.title(title)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()
```

```
visualize_loss(history, "Training and Validation Loss")
```

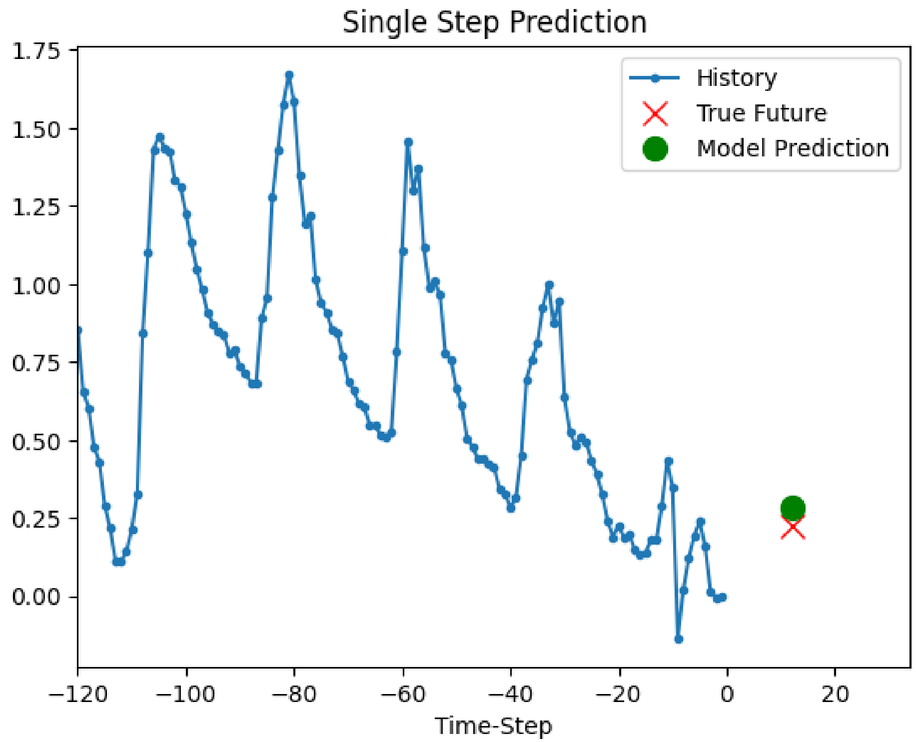


Step 8: Forecasting Future

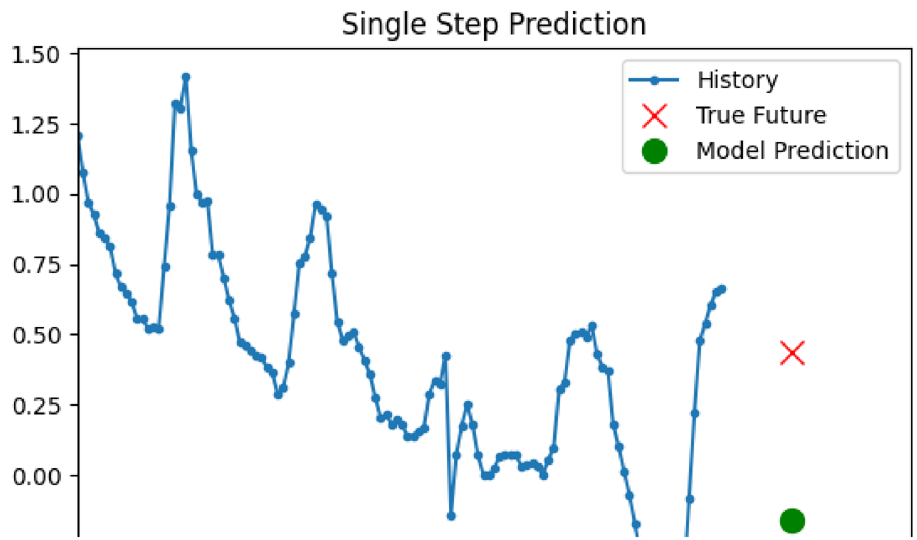
```
def show_plot(plot_data, delta, title):
    trueval=float(plot_data[1])
    pred=float(plot_data[2])
    err=round(((abs(pred-trueval)/trueval)*100),2)
    print("Error: "+str(err)+" %")
    labels = ["History", "True Future", "Model Prediction"]
    marker = [".-", "rx", "go"]
    time_steps = list(range(-(plot_data[0].shape[0]), 0))
    if delta:
        future = delta
    else:
        future = 0
    plt.title(title)
    for i, val in enumerate(plot_data):
        if i:
            plt.plot(future, plot_data[i], marker[i], markersize=10, label=labels[i])
        else:
           plt.plot(time_steps, plot_data[i].flatten(), marker[i], label=labels[i])
    plt.legend()
    plt.xlim([time_steps[0], (future + 5) * 2])
    plt.xlabel("Time-Step")
    plt.show()
    return
```

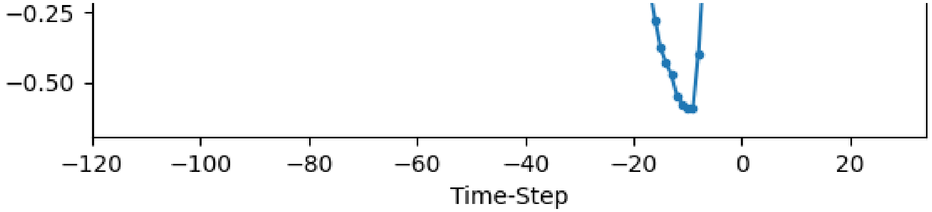
```
for x, y in dataset_val.take(10):
   show_plot(
      [x[0][:, 1].numpy(), y[0].numpy(), model.predict(x)[0]],
      12,
      "Single Step Prediction",
      )
```

```
8/8 [==============================] - 0s 31ms/step
Error: 25.49 %
```

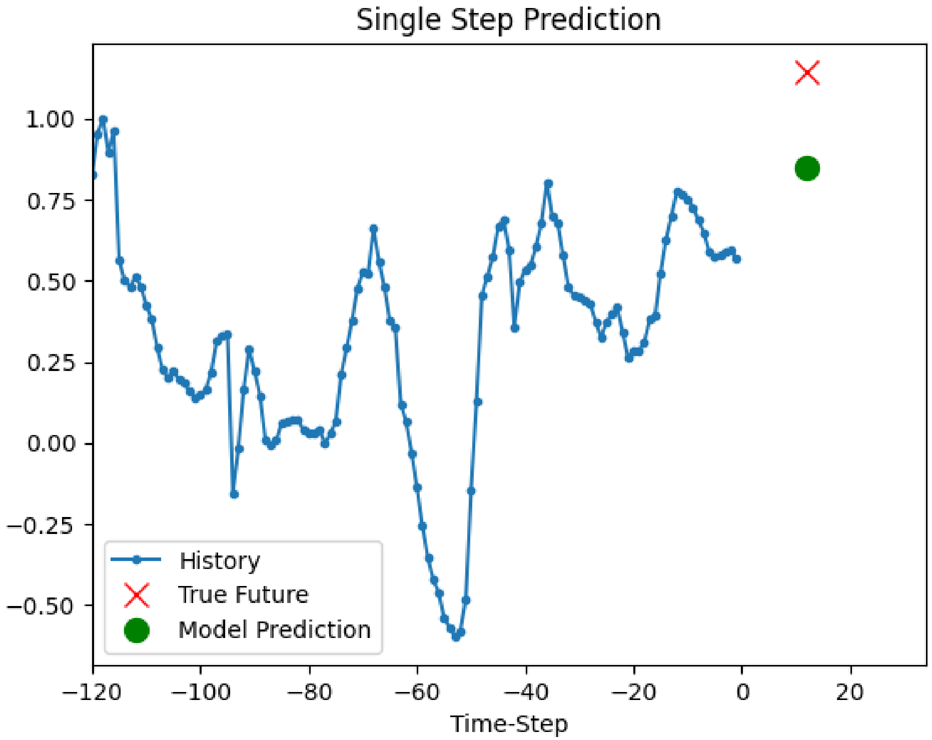



```
8/8 [==============================] - 0s 4ms/step
Error: 137.02 %
```

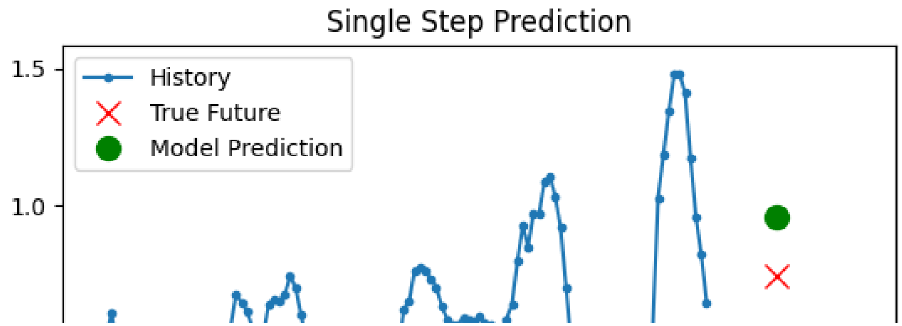


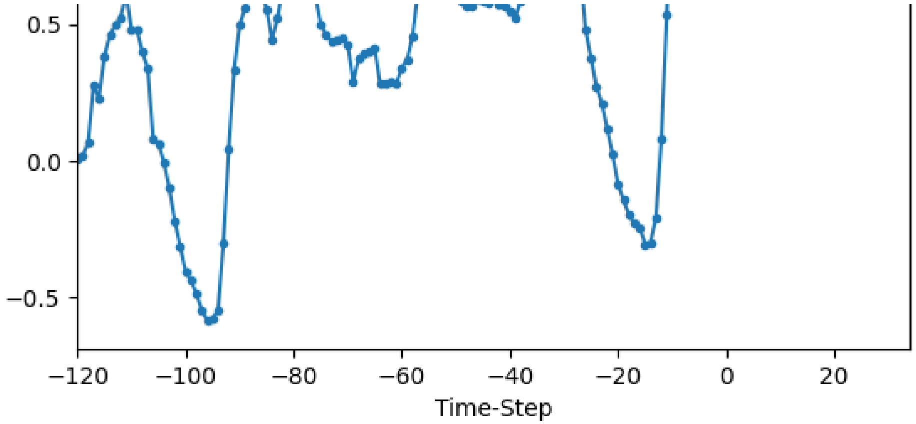



```
8/8 [==============================] - 0s 4ms/step
Error: 25.7 %
```

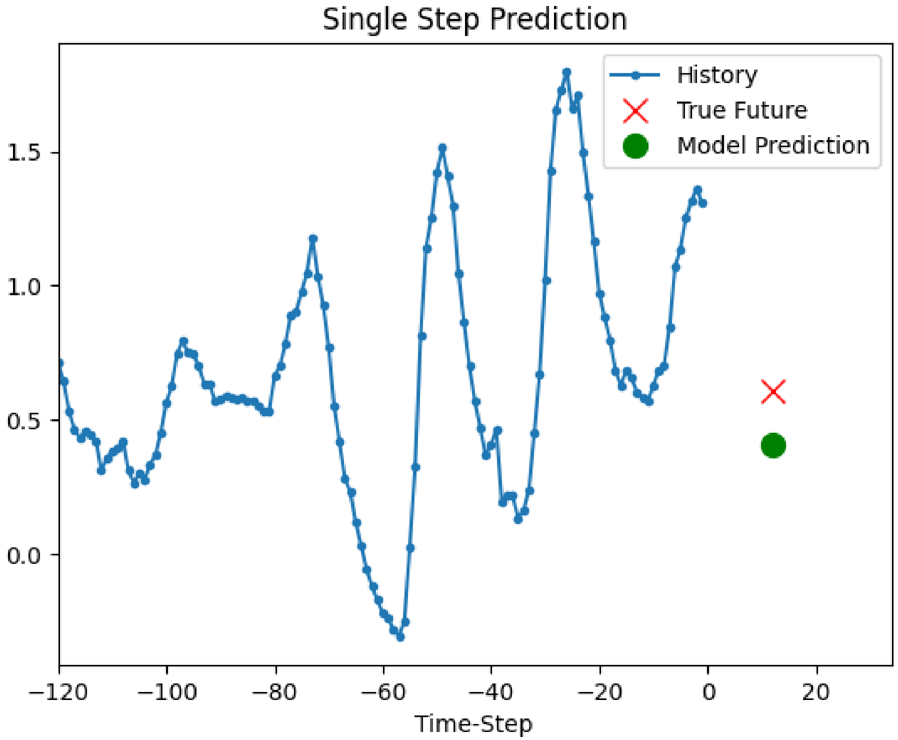



```
8/8 [==============================] - 0s 5ms/step
Error: 29.21 %
```

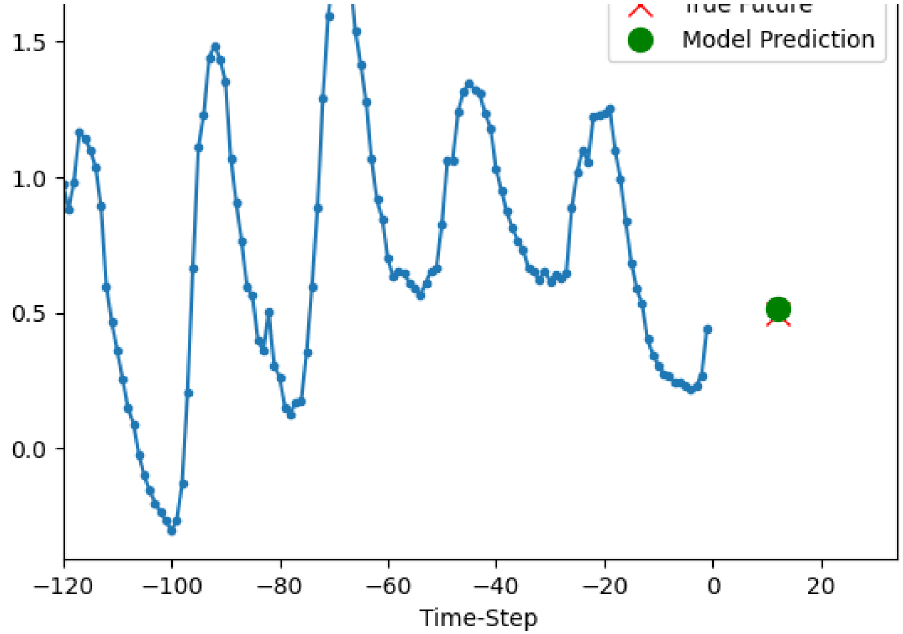





```
8/8 [==============================] - 0s 5ms/step
Error: 32.63 %
```

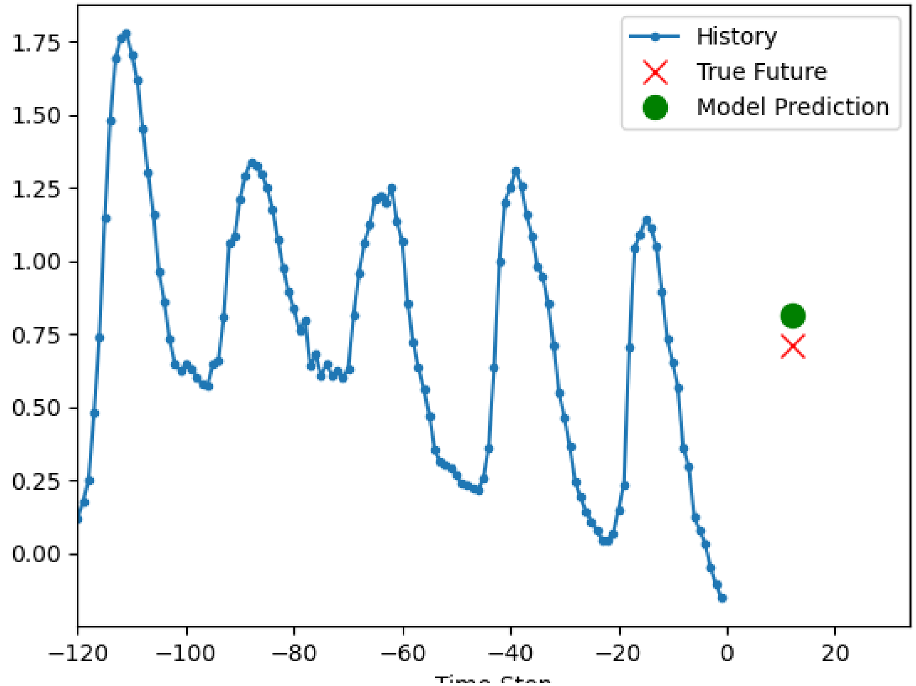



```
8/8 [==============================] - 0s 45ms/step
Error: 2.89 %
```



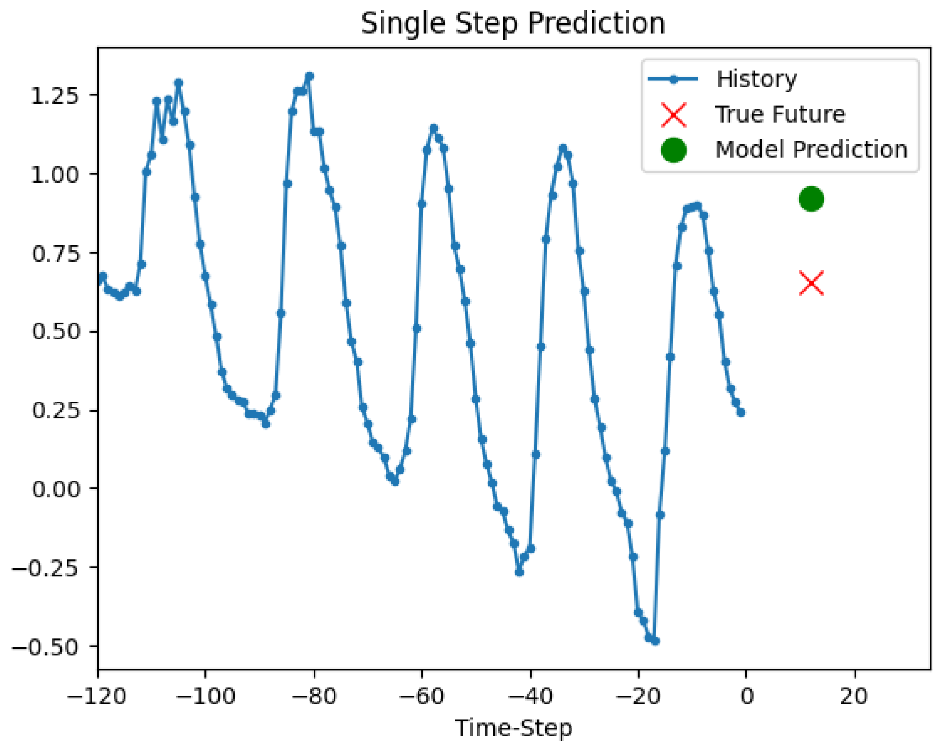

```
8/8 [==============================] - 0s 5ms/step
Error: 14.3 %
```

Single Step Prediction

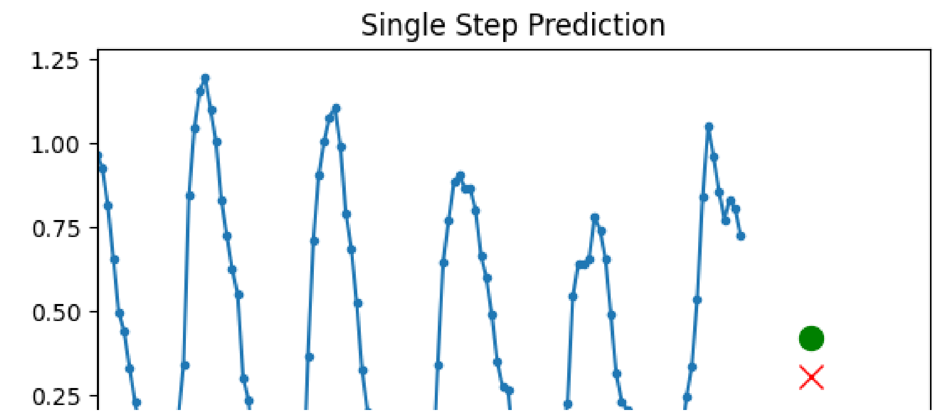


Time-Step

```
8/8 [==============================] - 0s 4ms/step
Error: 40.23 %
```




```
8/8 [==============================] - 0s 4ms/step
Error: 36.96 %
```



LAB_9.2

Aim: To build a custom attention layer to a deep learning network

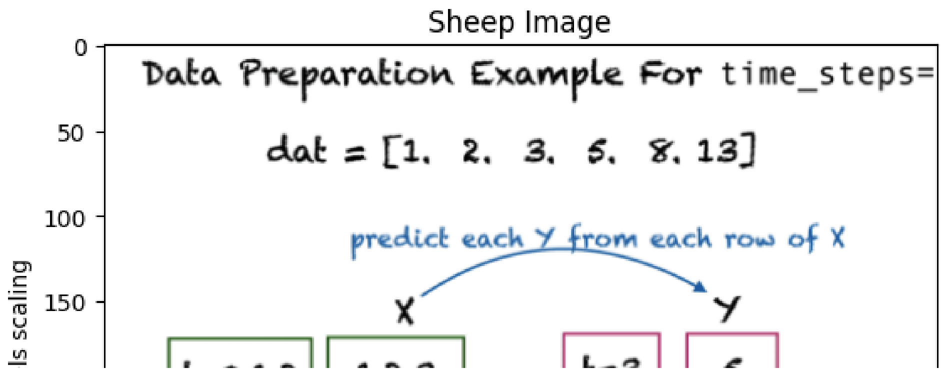
Let's use a very simple example of a Fibonacci sequence, where one number is constructed from the previous two numbers. The first 10 numbers of the sequence are shown below: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …When given the previous 't' numbers, can you get a machine to accurately reconstruct the next number? This would mean discarding all the previous inputs except the last two and performing the correct operation on the last two numbers.

Construct the training examples from t time steps and use the value at t+1 as the target. For example, if t=3, then the training examples and thecorresponding target values would look as follows:

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

plt.title("Sheep Image")
plt.xlabel("X pixel scaling")
plt.ylabel("Y pixels scaling")

image = mpimg.imread("DEEP.png")
plt.imshow(image)
plt.show()
```




```
import numpy as np
from keras import Model
from keras.layers import Layer
import keras.backend as K
from keras.layers import Input, Dense, SimpleRNN
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.metrics import mean_squared_error
```

0 100 200 300 400

PART I: Preparing the Dataset

Generate fibonoci series

```
def get_fib_seq(n, scale_data=True):
    # Get the Fibonacci sequence
    seq = np.zeros(n)
    fib_n1 = 0.0
    fib_n = 1.0
    for i in range(n):
            seq[i] = fib_n1 + fib_n
            fib_n1 = fib_n
            fib_n = seq[i]
    scaler = []
    if scale_data:
        scaler = MinMaxScaler(feature_range=(0, 1))
        seq = np.reshape(seq, (n, 1))
        seq = scaler.fit_transform(seq).flatten()
    return seq, scaler

fib_seq = get_fib_seq(10, False)[0]
```

```
fib_seq = get_fib_seq(10, False)[0]
print(fib_seq)
```

```
[ 1.  2.  3.  5.  8. 13. 21. 34. 55. 89.]
```

PART II: Training examples and Target values

```
def get_fib_XY(total_fib_numbers, time_steps, train_percent, scale_data=True):
    dat, scaler = get_fib_seq(total_fib_numbers, scale_data)
    Y_ind = np.arange(time_steps, len(dat), 1)
    Y = dat[Y_ind]
    rows_x = len(Y)
    X = dat[0:rows_x]
    for i in range(time_steps-1):
        temp = dat[i+1:rows_x+i+1]
        X = np.column_stack((X, temp))
    # random permutation with fixed seed
    rand = np.random.RandomState(seed=13)
    idx = rand.permutation(rows_x)
    split = int(train_percent*rows_x)
    train_ind = idx[0:split]
    test_ind = idx[split:]
    trainX = X[train_ind]
    trainY = Y[train_ind]
    testX = X[test_ind]
    testY = Y[test_ind]
    trainX = np.reshape(trainX, (len(trainX), time_steps, 1))
    testX = np.reshape(testX, (len(testX), time_steps, 1))
    return trainX, trainY, testX, testY, scaler

trainX, trainY, testX, testY, scaler = get_fib_XY(12, 3, 0.7, False)
print('trainX = ', trainX)
print('trainY = ', trainY)
```

```
trainX =  [[[ 8.]
  [13.]
  [21.]]

 [[ 5.]
  [ 8.]
  [13.]]

 [[ 2.]
```

```
  [ 3.]
  [ 5.]]

 [[13.]
  [21.]
  [34.]]

 [[21.]
  [34.]
  [55.]]

 [[34.]
  [55.]
  [89.]]]
trainY =  [ 34.  21.   8.  55.  89. 144.]
```

PART III: Setting Up the Network

Now let’s set up a small network with two layers. The first one is the SimpleRNN layer, and the second one is the Dense layer. Below is a summary of the model.

```
# Set up parameters
time_steps = 20
hidden_units = 2
epochs = 30

# Create a traditional RNN network
def create_RNN(hidden_units, dense_units, input_shape, activation):
    model = Sequential()
    model.add(SimpleRNN(hidden_units, input_shape=input_shape, activation=activation[0]))
    model.add(Dense(units=dense_units, activation=activation[1]))
    model.compile(loss='mse', optimizer='adam')
    return model

model_RNN = create_RNN(hidden_units=hidden_units, dense_units=1, input_shape=(time_steps,1),
                   activation=['tanh', 'tanh'])
model_RNN.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 simple_rnn_2 (SimpleRNN)    (None, 2)                 8

 dense_2 (Dense)             (None, 1)                 3

=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_________________________________________________________________
```

PART III: Train the Network and Evaluate

The next step is to add code that generates a dataset, trains the network, and evaluates it. This time around, we'll scale the data between 0 and 1. We don't need to pass the scale_data parameter as its default value is True.

```
# Generate the dataset
trainX, trainY, testX, testY, scaler  = get_fib_XY(1200, time_steps, 0.7)

model_RNN.fit(trainX, trainY, epochs=epochs, batch_size=1, verbose=2)
```

```
Epoch 1/30
826/826 - 5s - loss: 3.3553e-04 - 5s/epoch - 6ms/step
Epoch 2/30
826/826 - 2s - loss: 2.8307e-04 - 2s/epoch - 3ms/step
Epoch 3/30
826/826 - 2s - loss: 2.4739e-04 - 2s/epoch - 3ms/step
Epoch 4/30
826/826 - 2s - loss: 2.2280e-04 - 2s/epoch - 3ms/step
Epoch 5/30
826/826 - 3s - loss: 2.0338e-04 - 3s/epoch - 4ms/step
Epoch 6/30
826/826 - 3s - loss: 1.8223e-04 - 3s/epoch - 3ms/step
Epoch 7/30
826/826 - 2s - loss: 1.6502e-04 - 2s/epoch - 3ms/step
Epoch 8/30
826/826 - 2s - loss: 1.5173e-04 - 2s/epoch - 3ms/step
Epoch 9/30
826/826 - 2s - loss: 1.4242e-04 - 2s/epoch - 3ms/step
Epoch 10/30
826/826 - 3s - loss: 1.3393e-04 - 3s/epoch - 4ms/step
```

```
Epoch 11/30
826/826 - 3s - loss: 1.2237e-04 - 3s/epoch - 4ms/step
Epoch 12/30
826/826 - 2s - loss: 1.1836e-04 - 2s/epoch - 3ms/step
Epoch 13/30
826/826 - 2s - loss: 1.1156e-04 - 2s/epoch - 3ms/step
Epoch 14/30
826/826 - 3s - loss: 1.0175e-04 - 3s/epoch - 3ms/step
Epoch 15/30
826/826 - 3s - loss: 9.8370e-05 - 3s/epoch - 4ms/step
Epoch 16/30
826/826 - 3s - loss: 9.4179e-05 - 3s/epoch - 4ms/step
Epoch 17/30
826/826 - 2s - loss: 9.2336e-05 - 2s/epoch - 3ms/step
Epoch 18/30
826/826 - 2s - loss: 8.5180e-05 - 2s/epoch - 3ms/step
Epoch 19/30
826/826 - 2s - loss: 7.9596e-05 - 2s/epoch - 3ms/step
Epoch 20/30
826/826 - 2s - loss: 8.0695e-05 - 2s/epoch - 3ms/step
Epoch 21/30
826/826 - 4s - loss: 7.3901e-05 - 4s/epoch - 5ms/step
Epoch 22/30
826/826 - 2s - loss: 7.1297e-05 - 2s/epoch - 3ms/step
Epoch 23/30
826/826 - 2s - loss: 6.7814e-05 - 2s/epoch - 3ms/step
Epoch 24/30
826/826 - 2s - loss: 6.5458e-05 - 2s/epoch - 3ms/step
Epoch 25/30
826/826 - 2s - loss: 6.4373e-05 - 2s/epoch - 3ms/step
Epoch 26/30
826/826 - 4s - loss: 6.1040e-05 - 4s/epoch - 4ms/step
Epoch 27/30
826/826 - 3s - loss: 5.8691e-05 - 3s/epoch - 3ms/step
Epoch 28/30
826/826 - 2s - loss: 5.8074e-05 - 2s/epoch - 3ms/step
Epoch 29/30
826/826 - 2s - loss: 5.4346e-05 - 2s/epoch - 3ms/step
```

```
# Evalute model
train_mse = model_RNN.evaluate(trainX, trainY)
test_mse = model_RNN.evaluate(testX, testY)
```

```
26/26 [==============================] - 1s 3ms/step - loss: 4.0899e-05
12/12 [==============================] - 0s 4ms/step - loss: 7.6922e-06
```

```
# Print error
print("Train set MSE = ", train_mse)
```

```
print("Test set MSE = ", test_mse)

     Train set MSE =  4.089879075763747e-05
     Test set MSE =  7.692231520195492e-06
```

PART IV: Custom Attention Layer

```
# Add attention layer to the deep learning network
class attention(Layer):
    def __init__(self,**kwargs):
        super(attention,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(name='attention_weight', shape=(input_shape[-1],1),
                               initializer='random_normal', trainable=True)
        self.b=self.add_weight(name='attention_bias', shape=(input_shape[1],1),
                               initializer='zeros', trainable=True)
        super(attention, self).build(input_shape)

    def call(self,x):
        # Alignment scores. Pass them through tanh function
        e = K.tanh(K.dot(x,self.W)+self.b)
        # Remove dimension of size 1
        e = K.squeeze(e, axis=-1)
        # Compute the weights
        alpha = K.softmax(e)
        # Reshape to tensorFlow format
        alpha = K.expand_dims(alpha, axis=-1)
        # Compute the context vector
        context = x * alpha
        context = K.sum(context, axis=1)
        return context
```

PART V: RNN Network with Attention Layer

```
def create_RNN_with_attention(hidden_units, dense_units, input_shape, activation):
    x=Input(shape=input_shape)
    RNN_layer = SimpleRNN(hidden_units, return_sequences=True, activation=activation)(x)
```

```
    attention_layer = attention()(RNN_layer)
    outputs=Dense(dense_units, trainable=True, activation=activation)(attention_layer)
    model=Model(x,outputs)
    model.compile(loss='mse', optimizer='adam')
    return model

model_attention = create_RNN_with_attention(hidden_units=hidden_units, dense_units=1,
                                  input_shape=(time_steps,1), activation='tanh')
model_attention.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 20, 1)]           0

 simple_rnn_3 (SimpleRNN)    (None, 20, 2)             8

 attention (attention)       (None, 2)                 22

 dense_3 (Dense)             (None, 1)                 3

=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

Train and Evaluate the Deep Learning Network with Attention

```
model_attention.fit(trainX, trainY, epochs=epochs, batch_size=1, verbose=2)

# Evalute model
train_mse_attn = model_attention.evaluate(trainX, trainY)
test_mse_attn = model_attention.evaluate(testX, testY)

# Print error
print("Train set MSE with attention = ", train_mse_attn)
print("Test set MSE with attention = ", test_mse_attn)
```

```
Epoch 1/30
826/826 - 5s - loss: 0.0012 - 5s/epoch - 6ms/step
```

```
Epoch 2/30
826/826 - 3s - loss: 0.0012 - 3s/epoch - 3ms/step
Epoch 3/30
826/826 - 3s - loss: 0.0011 - 3s/epoch - 3ms/step
Epoch 4/30
826/826 - 4s - loss: 0.0011 - 4s/epoch - 5ms/step
Epoch 5/30
826/826 - 7s - loss: 0.0010 - 7s/epoch - 8ms/step
Epoch 6/30
826/826 - 5s - loss: 9.7444e-04 - 5s/epoch - 6ms/step
Epoch 7/30
826/826 - 3s - loss: 9.1638e-04 - 3s/epoch - 4ms/step
Epoch 8/30
826/826 - 4s - loss: 8.6438e-04 - 4s/epoch - 5ms/step
Epoch 9/30
826/826 - 3s - loss: 7.9930e-04 - 3s/epoch - 3ms/step
Epoch 10/30
826/826 - 3s - loss: 7.4029e-04 - 3s/epoch - 3ms/step
Epoch 11/30
826/826 - 3s - loss: 6.8214e-04 - 3s/epoch - 3ms/step
Epoch 12/30
826/826 - 3s - loss: 6.2416e-04 - 3s/epoch - 4ms/step
Epoch 13/30
826/826 - 4s - loss: 5.6176e-04 - 4s/epoch - 4ms/step
Epoch 14/30
826/826 - 3s - loss: 4.9927e-04 - 3s/epoch - 3ms/step
Epoch 15/30
826/826 - 3s - loss: 4.4027e-04 - 3s/epoch - 3ms/step
Epoch 16/30
826/826 - 3s - loss: 3.8568e-04 - 3s/epoch - 3ms/step
Epoch 17/30
```

Task: Build a custom layer in Simple RNN to predict the next lucas number

Lucas numbers are similar to Fibonacci numbers. Lucas numbers are also defined as the sum of its two immediately previous terms. But here the first two terms are 2 and 1 whereas in Fibonacci numbers the first two terms are 0 and 1 respectively. Mathematically, Lucas Numbers may be defined as: {\displaystyle L_{n}:={ }} The Lucas numbers are in the following integer sequence: 2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123 ………….

PART I: Preparing the Dataset

Generate Lucas series

```
def get_Lucas_seq(n, scale_data=True):
    # Get the Lucas sequence
    seq = np.zeros(n)
    Lucas_n1 = 2.0
    Lucas_n = 1.0
    for i in range(n):
            seq[i] = Lucas_n1 + Lucas_n
            Lucas_n1 = Lucas_n
            Lucas_n = seq[i]
    scaler = []
    if scale_data:
        scaler = MinMaxScaler(feature_range=(0, 1))
        seq = np.reshape(seq, (n, 1))
        seq = scaler.fit_transform(seq).flatten()
    return seq, scaler
Lucas_seq = get_Lucas_seq(300, False)[0]
print(Lucas_seq)
```

```
[3.00000000e+00 4.00000000e+00 7.00000000e+00 1.10000000e+01
 1.80000000e+01 2.90000000e+01 4.70000000e+01 7.60000000e+01
 1.23000000e+02 1.99000000e+02 3.22000000e+02 5.21000000e+02
 8.43000000e+02 1.36400000e+03 2.20700000e+03 3.57100000e+03
 5.77800000e+03 9.34900000e+03 1.51270000e+04 2.44760000e+04
 3.96030000e+04 6.40790000e+04 1.03682000e+05 1.67761000e+05
 2.71443000e+05 4.39204000e+05 7.10647000e+05 1.14985100e+06
 1.86049800e+06 3.01034900e+06 4.87084700e+06 7.88119600e+06
 1.27520430e+07 2.06332390e+07 3.33852820e+07 5.40185210e+07
 8.74038030e+07 1.41422324e+08 2.28826127e+08 3.70248451e+08
 5.99074578e+08 9.69323029e+08 1.56839761e+09 2.53772064e+09
 4.10611824e+09 6.64383888e+09 1.07499571e+10 1.73937960e+10
 2.81437531e+10 4.55375491e+10 7.36813022e+10 1.19218851e+11
 1.92900154e+11 3.12119005e+11 5.05019159e+11 8.17138164e+11
 1.32215732e+12 2.13929549e+12 3.46145281e+12 5.60074829e+12
 9.06220110e+12 1.46629494e+13 2.37251505e+13 3.83880999e+13
 6.21132504e+13 1.00501350e+14 1.62614601e+14 2.63115951e+14
 4.25730552e+14 6.88846503e+14 1.11457705e+15 1.80342356e+15
 2.91800061e+15 4.72142417e+15 7.63942478e+15 1.23608489e+16
 2.00002737e+16 3.23611227e+16 5.23613964e+16 8.47225191e+16
 1.37083915e+17 2.21806435e+17 3.58890350e+17 5.80696785e+17
 9.39587135e+17 1.52028392e+18 2.45987105e+18 3.98015497e+18
 6.44002603e+18 1.04201810e+19 1.68602070e+19 2.72803880e+19
 4.41405951e+19 7.14209831e+19 1.15561578e+20 1.86982561e+20
 3.02544139e+20 4.89526701e+20 7.92070840e+20 1.28159754e+21
 2.07366838e+21 3.35526592e+21 5.42893430e+21 8.78420022e+21
 1.42131345e+22 2.29973347e+22 3.72104693e+22 6.02078040e+22
 9.74182733e+22 1.57626077e+23 2.55044351e+23 4.12670428e+23
 6.67714778e+23 1.08038521e+24 1.74809998e+24 2.82848519e+24
```

```
 4.57658518e+24 7.40507037e+24 1.19816555e+25 1.93867259e+25
 3.13683815e+25 5.07551074e+25 8.21234888e+25 1.32878596e+26
 2.15002085e+26 3.47880681e+26 5.62882766e+26 9.10763447e+26
 1.47364621e+27 2.38440966e+27 3.85805587e+27 6.24246553e+27
 1.01005214e+28 1.63429869e+28 2.64435084e+28 4.27864953e+28
 6.92300036e+28 1.12016499e+29 1.81246503e+29 2.93263002e+29
 4.74509504e+29 7.67772506e+29 1.24228201e+30 2.01005452e+30
 3.25233653e+30 5.26239104e+30 8.51472757e+30 1.37771186e+31
 2.22918462e+31 3.60689648e+31 5.83608110e+31 9.44297757e+31
 1.52790587e+32 2.47220362e+32 4.00010949e+32 6.47231312e+32
 1.04724226e+33 1.69447357e+33 2.74171583e+33 4.43618940e+33
 7.17790524e+33 1.16140946e+34 1.87919999e+34 3.04060945e+34
 4.91980944e+34 7.96041889e+34 1.28802283e+35 2.08406472e+35
 3.37208756e+35 5.45615228e+35 8.82823983e+35 1.42843921e+36
 2.31126319e+36 3.73970241e+36 6.05096560e+36 9.79066801e+36
 1.58416336e+37 2.56323016e+37 4.14739352e+37 6.71062368e+37
 1.08580172e+38 1.75686409e+38 2.84266581e+38 4.59952990e+38
 7.44219571e+38 1.20417256e+39 1.94839213e+39 3.15256469e+39
 5.10095682e+39 8.25352152e+39 1.33544783e+40 2.16079999e+40
 3.49624782e+40 5.65704780e+40 9.15329562e+40 1.48103434e+41
 2.39636391e+41 3.87739825e+41 6.27376215e+41 1.01511604e+42
 1.64249226e+42 2.65760830e+42 4.30010055e+42 6.95770885e+42
 1.12578094e+43 1.82155182e+43 2.94733276e+43 4.76888459e+43
 7.71621735e+43 1.24851019e+44 2.02013193e+44 3.26864212e+44
 5.28877405e+44 8.55741618e+44 1.38461902e+45 2.24036064e+45
 3.62497966e+45 5.86534030e+45 9.49031997e+45 1.53556603e+46
 2.48459802e+46 4.02016405e+46 6.50476208e+46 1.05249261e+47
 1.70296882e+47 2.75546143e+47 4.45843025e+47 7.21389169e+47
 1.16723219e+48 1.88862136e+48 3.05585356e+48 4.94447492e+48
```

PART II: Training examples and Target values

```
def get_Lucas_XY(total_Lucas_numbers, time_steps, train_percent, scale_data=True):
    dat, scaler = get_Lucas_seq(total_Lucas_numbers, scale_data)
    Y_ind = np.arange(time_steps, len(dat), 1)
    Y = dat[Y_ind]
    rows_x = len(Y)
    X = dat[0:rows_x]
    for i in range(time_steps-1):
        temp = dat[i+1:rows_x+i+1]
        X = np.column_stack((X, temp))
    # random permutation with fixed seed
    rand = np.random.RandomState(seed=13)
    idx = rand.permutation(rows_x)
    split = int(train_percent*rows_x)
```

```
    train_ind = idx[0:split]
    test_ind = idx[split:]
    trainX = X[train_ind]
    trainY = Y[train_ind]
    testX = X[test_ind]
    testY = Y[test_ind]
    trainX = np.reshape(trainX, (len(trainX), time_steps, 1))
    testX = np.reshape(testX, (len(testX), time_steps, 1))
    return trainX, trainY, testX, testY, scaler

trainX, trainY, testX, testY, scaler = get_Lucas_XY(12, 3, 0.7, False)
print('trainX = ', trainX)
print('trainY = ', trainY)
```

```
trainX =  [[[ 18.]
  [ 29.]
  [ 47.]]

 [[ 11.]
  [ 18.]
  [ 29.]]

 [[  4.]
  [  7.]
  [ 11.]]

 [[ 29.]
  [ 47.]
  [ 76.]]

 [[ 47.]
  [ 76.]
  [123.]]

 [[ 76.]
  [123.]
  [199.]]]
trainY =  [ 76.  47.  18. 123. 199. 322.]
```

PART III: Setting Up the Network

Now let’s set up a small network with two layers. The first one is the SimpleRNN layer, and the second one is the Dense layer. Below is a summary of the model.

```
# Set up parameters
time_steps = 20
hidden_units = 2
epochs = 30

# Create a traditional RNN network
def create_RNN(hidden_units, dense_units, input_shape, activation):
    model = Sequential()
    model.add(SimpleRNN(hidden_units, input_shape=input_shape, activation=activation[0]))
    model.add(Dense(units=dense_units, activation=activation[1]))
    model.compile(loss='mse', optimizer='adam')
    return model

model_RNN = create_RNN(hidden_units=hidden_units, dense_units=1, input_shape=(time_steps,1),
                   activation=['tanh', 'tanh'])
model_RNN.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_6 (SimpleRNN)    (None, 2)                 8

 dense_6 (Dense)             (None, 1)                 3

=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_________________________________________________________________
```

PART III: Train the Network and Evaluate

The next step is to add code that generates a dataset, trains the network, and evaluates it. This time around, we'll scale the data between 0 and 1. We don't need to pass the scale_data parameter as its default value is True.

```
# Generate the dataset
trainX, trainY, testX, testY, scaler  = get_Lucas_XY(1200, time_steps, 0.7)
model_RNN.fit(trainX, trainY, epochs=epochs, batch_size=1, verbose=2)
```

```
Epoch 1/30
826/826 - 5s - loss: 0.0016 - 5s/epoch - 6ms/step
Epoch 2/30
826/826 - 3s - loss: 0.0015 - 3s/epoch - 4ms/step
Epoch 3/30
826/826 - 3s - loss: 0.0015 - 3s/epoch - 3ms/step
Epoch 4/30
826/826 - 3s - loss: 0.0015 - 3s/epoch - 3ms/step
Epoch 5/30
826/826 - 2s - loss: 0.0014 - 2s/epoch - 3ms/step
Epoch 6/30
826/826 - 4s - loss: 0.0014 - 4s/epoch - 4ms/step
Epoch 7/30
826/826 - 3s - loss: 0.0014 - 3s/epoch - 4ms/step
Epoch 8/30
826/826 - 2s - loss: 0.0013 - 2s/epoch - 3ms/step
Epoch 9/30
826/826 - 2s - loss: 0.0013 - 2s/epoch - 3ms/step
Epoch 10/30
826/826 - 2s - loss: 0.0012 - 2s/epoch - 3ms/step
Epoch 11/30
826/826 - 3s - loss: 0.0011 - 3s/epoch - 4ms/step
Epoch 12/30
826/826 - 3s - loss: 0.0011 - 3s/epoch - 4ms/step
Epoch 13/30
826/826 - 2s - loss: 9.5463e-04 - 2s/epoch - 3ms/step
Epoch 14/30
826/826 - 2s - loss: 8.7171e-04 - 2s/epoch - 3ms/step
Epoch 15/30
826/826 - 2s - loss: 7.6627e-04 - 2s/epoch - 3ms/step
Epoch 16/30
826/826 - 3s - loss: 6.6340e-04 - 3s/epoch - 4ms/step
Epoch 17/30
826/826 - 3s - loss: 5.6273e-04 - 3s/epoch - 4ms/step
Epoch 18/30
826/826 - 2s - loss: 4.8612e-04 - 2s/epoch - 3ms/step
Epoch 19/30
826/826 - 2s - loss: 3.9764e-04 - 2s/epoch - 3ms/step
Epoch 20/30
826/826 - 2s - loss: 3.2416e-04 - 2s/epoch - 3ms/step
Epoch 21/30
826/826 - 3s - loss: 2.5358e-04 - 3s/epoch - 4ms/step
Epoch 22/30
826/826 - 3s - loss: 2.0115e-04 - 3s/epoch - 4ms/step
Epoch 23/30
826/826 - 2s - loss: 1.5177e-04 - 2s/epoch - 3ms/step
Epoch 24/30
826/826 - 2s - loss: 1.1829e-04 - 2s/epoch - 3ms/step
```

```
Epoch 25/30
826/826 - 2s - loss: 9.6721e-05 - 2s/epoch - 3ms/step
Epoch 26/30
826/826 - 2s - loss: 7.9366e-05 - 2s/epoch - 3ms/step
Epoch 27/30
826/826 - 4s - loss: 6.5789e-05 - 4s/epoch - 5ms/step
Epoch 28/30
826/826 - 2s - loss: 6.4677e-05 - 2s/epoch - 3ms/step
Epoch 29/30
```

```
# Evalute model
train_mse = model_RNN.evaluate(trainX, trainY)
test_mse = model_RNN.evaluate(testX, testY)
```

```
26/26 [==============================] - 0s 3ms/step - loss: 3.8801e-05
12/12 [==============================] - 0s 3ms/step - loss: 5.9619e-07
```

```
# Print error
print("Train set MSE = ", train_mse)
print("Test set MSE = ", test_mse)
```

```
Train set MSE =  3.88005719287321e-05
Test set MSE =  5.961869078419113e-07
```

PART IV: Custom Attention Layer

```
# Add attention layer to the deep learning network
class attention(Layer):
    def __init__(self,**kwargs):
        super(attention,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(name='attention_weight', shape=(input_shape[-1],1),
                               initializer='random_normal', trainable=True)
        self.b=self.add_weight(name='attention_bias', shape=(input_shape[1],1),
                               initializer='zeros', trainable=True)
        super(attention, self).build(input_shape)

    def call(self,x):
        # Alignment scores. Pass them through tanh function
        e = K.tanh(K.dot(x,self.W)+self.b)
        # Remove dimension of size 1
```

```
        e = K.squeeze(e, axis=-1)
        # Compute the weights
        alpha = K.softmax(e)
        # Reshape to tensorFlow format
        alpha = K.expand_dims(alpha, axis=-1)
        # Compute the context vector
        context = x * alpha
        context = K.sum(context, axis=1)
        return context
```

PART V: RNN Network with Attention Layer

```
def create_RNN_with_attention(hidden_units, dense_units, input_shape, activation):
    x=Input(shape=input_shape)
    RNN_layer = SimpleRNN(hidden_units, return_sequences=True, activation=activation)(x)
    attention_layer = attention()(RNN_layer)
    outputs=Dense(dense_units, trainable=True, activation=activation)(attention_layer)
    model=Model(x,outputs)
    model.compile(loss='mse', optimizer='adam')
    return model

model_attention = create_RNN_with_attention(hidden_units=hidden_units, dense_units=1,
                                  input_shape=(time_steps,1), activation='tanh')
model_attention.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 20, 1)]           0

 simple_rnn_7 (SimpleRNN)    (None, 20, 2)             8

 attention_2 (attention)     (None, 2)                 22

 dense_7 (Dense)             (None, 1)                 3

=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

Train and Evaluate the Deep Learning Network with Attention

```
model_attention.fit(trainX, trainY, epochs=epochs, batch_size=1, verbose=2)

# Evalute model
train_mse_attn = model_attention.evaluate(trainX, trainY)
test_mse_attn = model_attention.evaluate(testX, testY)

# Print error
print("Train set MSE with attention = ", train_mse_attn)
print("Test set MSE with attention = ", test_mse_attn)
```

```
Epoch 1/30
826/826 - 5s - loss: 0.0014 - 5s/epoch - 6ms/step
Epoch 2/30
826/826 - 3s - loss: 0.0014 - 3s/epoch - 3ms/step
Epoch 3/30
826/826 - 3s - loss: 0.0013 - 3s/epoch - 4ms/step
Epoch 4/30
826/826 - 4s - loss: 0.0013 - 4s/epoch - 5ms/step
Epoch 5/30
826/826 - 3s - loss: 0.0013 - 3s/epoch - 4ms/step
Epoch 6/30
826/826 - 3s - loss: 0.0012 - 3s/epoch - 4ms/step
Epoch 7/30
826/826 - 3s - loss: 0.0012 - 3s/epoch - 4ms/step
Epoch 8/30
826/826 - 4s - loss: 0.0012 - 4s/epoch - 5ms/step
Epoch 9/30
826/826 - 3s - loss: 0.0011 - 3s/epoch - 4ms/step
Epoch 10/30
826/826 - 3s - loss: 0.0011 - 3s/epoch - 4ms/step
Epoch 11/30
826/826 - 3s - loss: 9.8820e-04 - 3s/epoch - 4ms/step
Epoch 12/30
826/826 - 4s - loss: 9.3905e-04 - 4s/epoch - 5ms/step
Epoch 13/30
826/826 - 4s - loss: 8.9244e-04 - 4s/epoch - 5ms/step
Epoch 14/30
826/826 - 3s - loss: 8.3749e-04 - 3s/epoch - 4ms/step
Epoch 15/30
826/826 - 3s - loss: 7.8253e-04 - 3s/epoch - 4ms/step
Epoch 16/30
```

```
826/826 - 4s - loss: 7.1790e-04 - 4s/epoch - 5ms/step
Epoch 17/30
826/826 - 4s - loss: 6.6568e-04 - 4s/epoch - 5ms/step
Epoch 18/30
826/826 - 3s - loss: 6.0930e-04 - 3s/epoch - 4ms/step
Epoch 19/30
826/826 - 3s - loss: 5.5676e-04 - 3s/epoch - 4ms/step
Epoch 20/30
826/826 - 4s - loss: 5.0641e-04 - 4s/epoch - 5ms/step
Epoch 21/30
826/826 - 3s - loss: 4.6250e-04 - 3s/epoch - 4ms/step
Epoch 22/30
826/826 - 3s - loss: 4.1928e-04 - 3s/epoch - 4ms/step
Epoch 23/30
826/826 - 3s - loss: 3.8001e-04 - 3s/epoch - 4ms/step
Epoch 24/30
826/826 - 4s - loss: 3.3669e-04 - 4s/epoch - 5ms/step
Epoch 25/30
826/826 - 4s - loss: 2.9210e-04 - 4s/epoch - 4ms/step
Epoch 26/30
826/826 - 3s - loss: 2.5745e-04 - 3s/epoch - 4ms/step
Epoch 27/30
826/826 - 3s - loss: 2.2902e-04 - 3s/epoch - 4ms/step
Epoch 28/30
826/826 - 3s - loss: 2.0710e-04 - 3s/epoch - 4ms/step
Epoch 29/30
```