In [1]:

```python
#imports

import tensorflow as tf

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

# gloabl params for all matplotlib plots
mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False
```

In [2]:

```python
df=pd.read_csv("Bangalore_Climate_Data.csv")
df.head()
```

Out[2]:

	Date Time	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)
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.3	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
1	01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.4	3.23	3.02	0.21	1.89	3.03	1309.80	0.72	1.50	136.1
2	01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.9	3.21	3.01	0.20	1.88	3.02	1310.24	0.19	0.63	171.6
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.2	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.50	198.0
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.1	3.27	3.08	0.19	1.92	3.09	1309.00	0.32	0.63	214.3

In [3]:

```python
df.describe()
```

Out[3]:

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H (mmol/

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/
count	420551.000000	420551.000000	420551.000000	420551.000000	420551.000000	420551.000000	420551.000000	420551.000000	420551.000000	420551.00
mean	989.212776	9.450147	283.492743	4.955854	76.008259	13.576251	9.533756	4.042412	6.022408	9.64
std	8.358481	8.423365	8.504471	6.730674	16.476175	7.739020	4.184164	4.896851	2.656139	4.23
min	913.600000	-23.010000	250.600000	-25.010000	12.950000	0.950000	0.790000	0.000000	0.500000	0.80
25%	984.200000	3.360000	277.430000	0.240000	65.210000	7.780000	6.210000	0.870000	3.920000	6.29
50%	989.580000	9.420000	283.470000	5.220000	79.300000	11.820000	8.860000	2.190000	5.590000	8.96
75%	994.720000	15.470000	289.530000	10.070000	89.400000	17.600000	12.350000	5.300000	7.800000	12.49
max	1015.350000	37.280000	311.340000	23.110000	100.000000	63.770000	28.320000	46.010000	18.130000	28.82

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 420551 entries, 0 to 420550
Data columns (total 15 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   Date Time        420551 non-null  object 
 1   p (mbar)         420551 non-null  float64
 2   T (degC)         420551 non-null  float64
 3   Tpot (K)         420551 non-null  float64
 4   Tdew (degC)      420551 non-null  float64
 5   rh (%)           420551 non-null  float64
 6   VPmax (mbar)     420551 non-null  float64
 7   VPact (mbar)     420551 non-null  float64
 8   VPdef (mbar)     420551 non-null  float64
 9   sh (g/kg)        420551 non-null  float64
 10  H2OC (mmol/mol)  420551 non-null  float64
 11  rho (g/m**3)     420551 non-null  float64
 12  wv (m/s)         420551 non-null  float64
 13  max. wv (m/s)    420551 non-null  float64
 14  wd (deg)         420551 non-null  float64
dtypes: float64(14), object(1)
memory usage: 48.1+ MB
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Date Time          0
p (mbar)           0
T (degC)           0
Tpot (K)           0
Tdew (degC)        0
rh (%)             0
VPmax (mbar)       0
VPact (mbar)       0
VPdef (mbar)       0
sh (g/kg)          0
H2OC (mmol/mol)    0
rho (g/m**3)       0
wv (m/s)           0
max. wv (m/s)      0
wd (deg)           0
dtype: int64
```

Observations:

1. One reading every 10 mins
2. 1 day = 6*24 = 144 readings
3. 5 days = 144*5 = 720 readings

Forecasting task: Predict temperature (in deg C) in the future.

In [6]:
```
# univariate data: Temp vs Time

uni_data_df = df['T (degC)']
uni_data_df.index = df['Date Time']
uni_data_df.index = pd.to_datetime(uni_data_df.index)
uni_data_df.head()
```

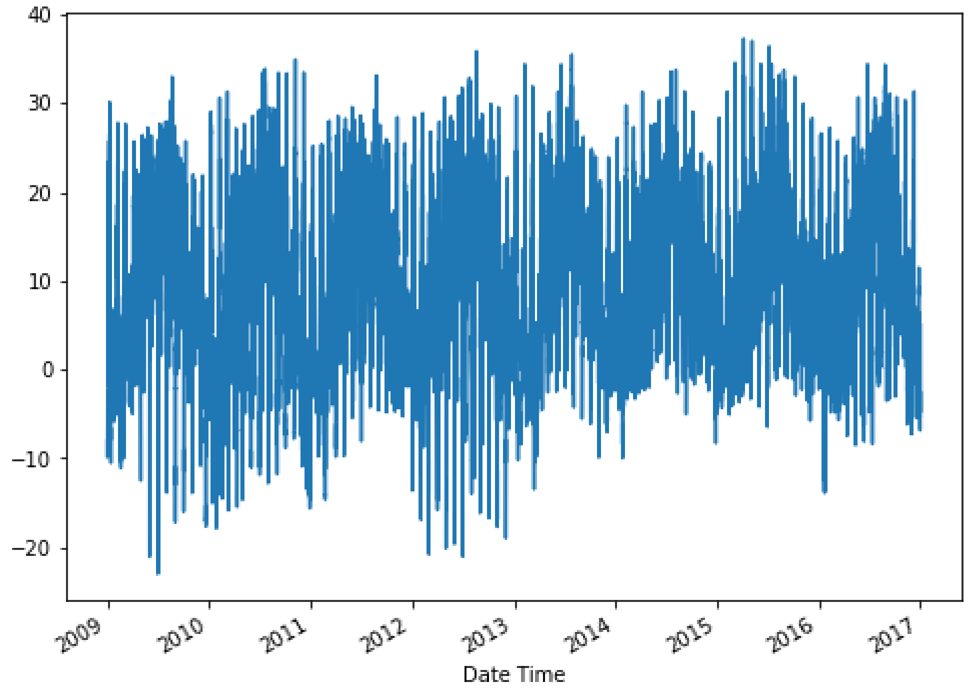
Out[6]:
```
Date Time
2009-01-01 00:10:00   -8.02
2009-01-01 00:20:00   -8.41
2009-01-01 00:30:00   -8.51
2009-01-01 00:40:00   -8.31
2009-01-01 00:50:00   -8.27
Name: T (degC), dtype: float64
```

In [7]:

```
uni_data_df.plot()
```

Out[7]:

```
<AxesSubplot:xlabel='Date Time'>
```



In [8]:

```
uni_data = uni_data_df.values # numpy ndarray from pandas
```

In [9]:

```
TRAIN_SPLIT = 300000 # First 300000 obs will be used as train data and rest as test data.
# 300,000 => ~2100 days worth of training data


tf.random.set_seed(13) # random seed

# Normalize data: mean centering and variance-scaling.
# NOTE: use only train data to normalize all of the data. otherwise, leakage-issue
uni_train_mean = uni_data[:TRAIN_SPLIT].mean()
uni_train_std = uni_data[:TRAIN_SPLIT].std()
```

```python
uni_data = (uni_data-uni_train_mean)/uni_train_std
print(type(uni_data))
```

```
<class 'numpy.ndarray'>
```

Moving window average

Pose a simple problem:

Given last 'k' values of temp-observations (only one feature <=> univariate), predict the next observation

MWA:

Average the previous k values to predict the next value.

In [10]:

```python
# This function creates the data we need for the above problem
# dataset: numpy ndarray
# start_index:
# end_index:
# history_size: k => take k values at a time
# target_size: 0 => next value in the time-series
# Output: data: (n,k) and labels (n,1)

def univariate_data(dataset, start_index, end_index, history_size, target_size):
  data = []
  labels = []

  start_index = start_index + history_size
  if end_index is None:
    end_index = len(dataset) - target_size

  print(start_index, end_index)
  for i in range(start_index, end_index):
    indices = range(i-history_size, i)
    # Reshape data from (history_size,) to (history_size, 1)
    data.append(np.reshape(dataset[indices], (history_size, 1)))
    labels.append(dataset[i+target_size])
  return np.array(data), np.array(labels)
```

```
# use the above function to create the datasets.
univariate_past_history = 20
univariate_future_target = 0

x_train_uni, y_train_uni = univariate_data(uni_data, 0, TRAIN_SPLIT,
                                           univariate_past_history,
                                           univariate_future_target)
x_val_uni, y_val_uni = univariate_data(uni_data, TRAIN_SPLIT, None,
                                       univariate_past_history,
                                       univariate_future_target)

print(x_train_uni.shape)
print(y_train_uni.shape)
print(x_val_uni.shape)
print(y_val_uni.shape)
```

```
20 300000
300020 420551
(299980, 20, 1)
(299980,)
(120531, 20, 1)
(120531,)
```

In [11]:

```
np.reshape(uni_data[range(1,20)], (19,1))
```

Out[11]:

```
array([[-2.04281897],
       [-2.05439744],
       [-2.0312405 ],
       [-2.02660912],
       [-2.00113649],
       [-1.95134907],
       [-1.95134907],
       [-1.98492663],
       [-2.04513467],
       [-2.08334362],
       [-2.09723778],
       [-2.09376424],
       [-2.09144854],
       [-2.07176515],
       [-2.07176515],
       [-2.07639653],
       [-2.08913285],
```

```
       [-2.09260639],
       [-2.10418486]])
```

In [12]:

```
print ('Single window of past history')
print (x_train_uni[0])
print ('\n Target temperature to predict')
print (y_train_uni[0])
```

```
Single window of past history
[[-1.99766294]
 [-2.04281897]
 [-2.05439744]
 [-2.0312405 ]
 [-2.02660912]
 [-2.00113649]
 [-1.95134907]
 [-1.95134907]
 [-1.98492663]
 [-2.04513467]
 [-2.08334362]
 [-2.09723778]
 [-2.09376424]
 [-2.09144854]
 [-2.07176515]
 [-2.07176515]
 [-2.07639653]
 [-2.08913285]
 [-2.09260639]
 [-2.10418486]]

 Target temperature to predict
-2.1041848598100876
```

In [13]:

```
#utility function
def create_time_steps(length):
  return list(range(-length, 0))

print(create_time_steps(20))
```

```
[-20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]
```

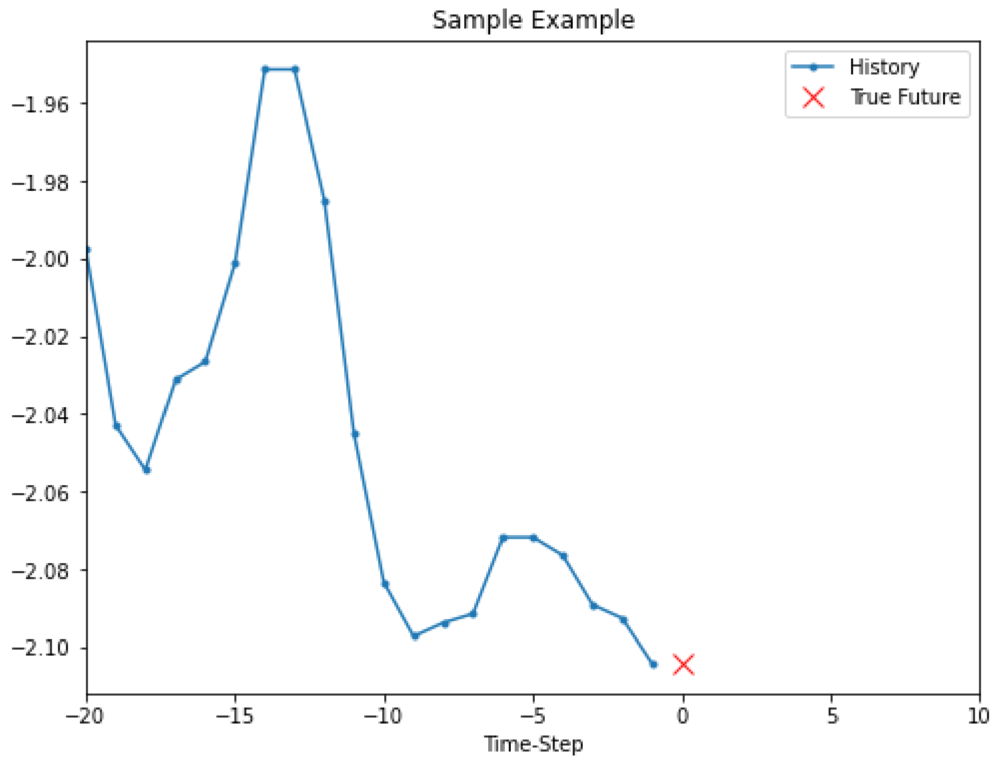
In [14]:

```
# Plotting function
```

```python
# plot_data: contains labels as list
# delta: 0 => next time step given last "k" steps.
# title: plot title

# Usage: show_plot([x_train_uni[0], y_train_uni[0]], 0, 'Sample Example')

def show_plot(plot_data, delta, title):
  labels = ['History', 'True Future', 'Model Prediction']
  marker = ['.-', 'rx', 'go'] # dot-line, red-x, green-o  refer: https://matplotlib.org/3.1.1/api/markers_api.html
  time_steps = create_time_steps(plot_data[0].shape[0])

  if delta:
    future = delta
  else:
    future = 0

  plt.title(title)
  for i, x 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')
  return plt


show_plot([x_train_uni[0], y_train_uni[0]], 0, 'Sample Example')
```

Out[14]:

```
<module 'matplotlib.pyplot' from 'D:\\ankonda\\lib\\site-packages\\matplotlib\\pyplot.py'>
```

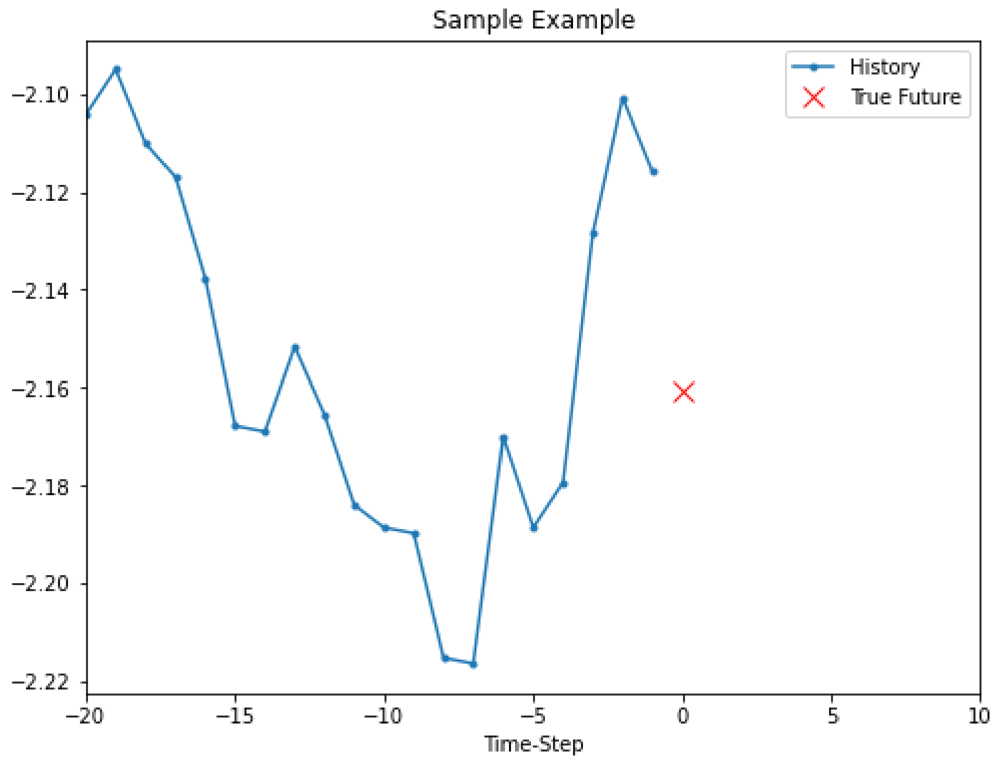


In [15]:

```
i=20
show_plot([x_train_uni[i], y_train_uni[i]], 0, 'Sample Example')
```

Out[15]:

```
<module 'matplotlib.pyplot' from 'D:\\ankonda\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



In [16]:

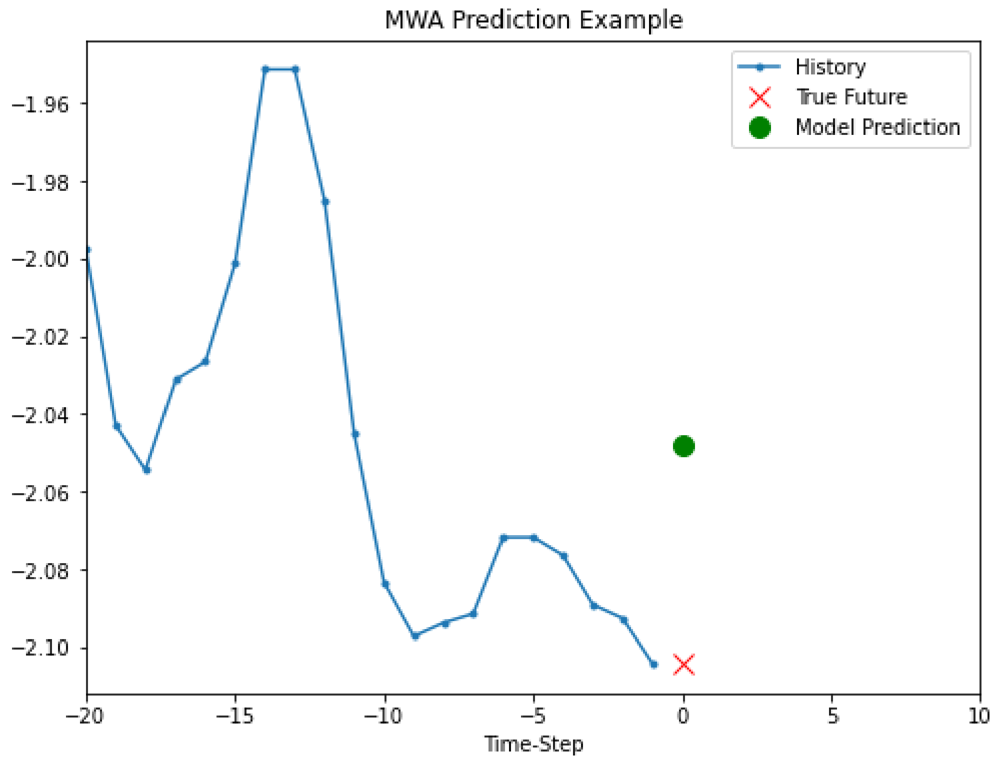
```
def mwa(history):
  return np.mean(history)
```

In [17]:

```
i=0
show_plot([x_train_uni[i], y_train_uni[i], mwa(x_train_uni[i])], 0,
           'MWA Prediction Example')
```

Out[17]:

```
<module 'matplotlib.pyplot' from 'D:\\ankonda\\lib\\site-packages\\matplotlib\\pyplot.py'>
```

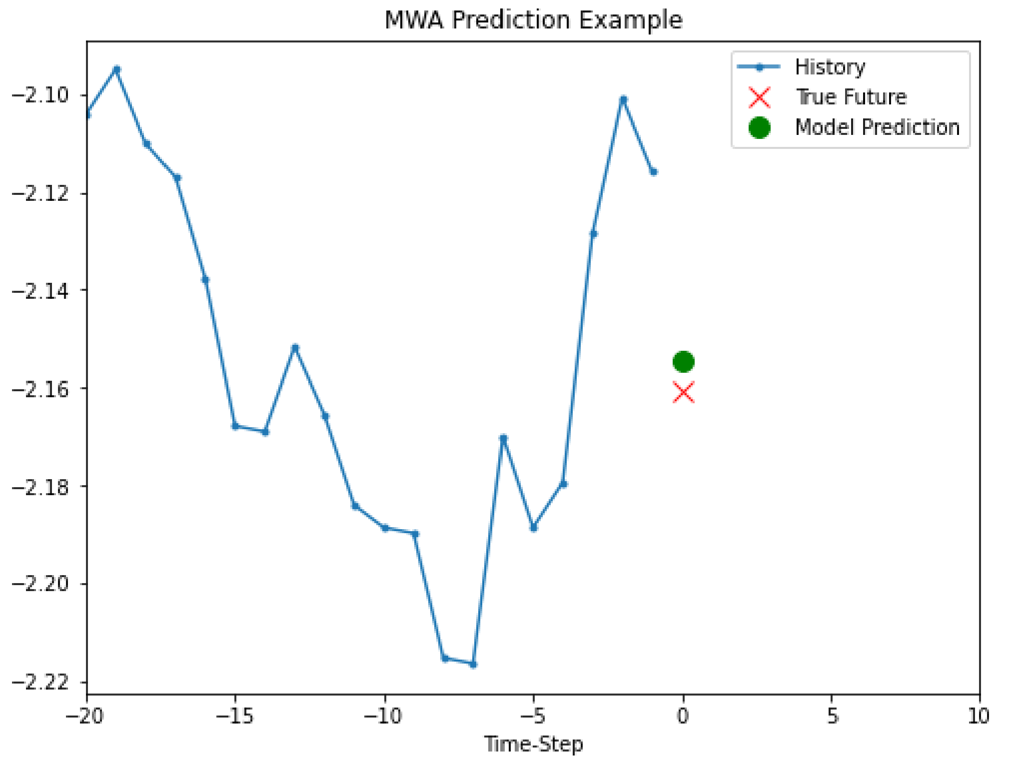


In [18]:

```
i=20
show_plot([x_train_uni[i], y_train_uni[i], mwa(x_train_uni[i])], 0,
            'MWA Prediction Example')
```

Out[18]:

```
<module 'matplotlib.pyplot' from 'D:\\ankonda\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



Univariate time-series forecasting

- Features from the history: only temperature => univariate
- Problem definition: Given last "k=20" values of temp, predict the next temp value.

In [19]:

```
# TF Dataset preperation
BATCH_SIZE = 256 # bacth size in batch-SGD/variants
BUFFER_SIZE = 10000 # for shuffling the dataset

train_univariate = tf.data.Dataset.from_tensor_slices((x_train_uni, y_train_uni))
train_univariate = train_univariate.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()


val_univariate = tf.data.Dataset.from_tensor_slices((x_val_uni, y_val_uni))
```

```python
val_univariate = val_univariate.batch(BATCH_SIZE).repeat()

print(train_univariate)
print(val_univariate)
```

```
<RepeatDataset element_spec=(TensorSpec(shape=(None, 20, 1), dtype=tf.float64, name=None), TensorSpec(shape=(None,), dtype=tf.floa
t64, name=None))>
<RepeatDataset element_spec=(TensorSpec(shape=(None, 20, 1), dtype=tf.float64, name=None), TensorSpec(shape=(None,), dtype=tf.floa
t64, name=None))>
```

In [20]:

```python
# MODEL:
simple_lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(8, input_shape=x_train_uni.shape[-2:]),
    tf.keras.layers.Dense(1)
])

simple_lstm_model.compile(optimizer='adam', loss='mae')
```

In [21]:

```python
# Train and evaluate
STEPS_PER_EPOCH = 200
EPOCHS = 10

simple_lstm_model.fit(train_univariate, epochs=EPOCHS,
                      steps_per_epoch=STEPS_PER_EPOCH,
                      validation_data=val_univariate, validation_steps=50)
```

```
Epoch 1/10
200/200 [==============================] - 7s 20ms/step - loss: 0.4075 - val_loss: 0.1351
Epoch 2/10
200/200 [==============================] - 3s 17ms/step - loss: 0.1118 - val_loss: 0.0359
Epoch 3/10
200/200 [==============================] - 3s 17ms/step - loss: 0.0489 - val_loss: 0.0290
Epoch 4/10
200/200 [==============================] - 3s 16ms/step - loss: 0.0443 - val_loss: 0.0258
Epoch 5/10
200/200 [==============================] - 3s 16ms/step - loss: 0.0299 - val_loss: 0.0235
Epoch 6/10
200/200 [==============================] - 3s 16ms/step - loss: 0.0317 - val_loss: 0.0224
Epoch 7/10
200/200 [==============================] - 3s 17ms/step - loss: 0.0286 - val_loss: 0.0207
Epoch 8/10
```

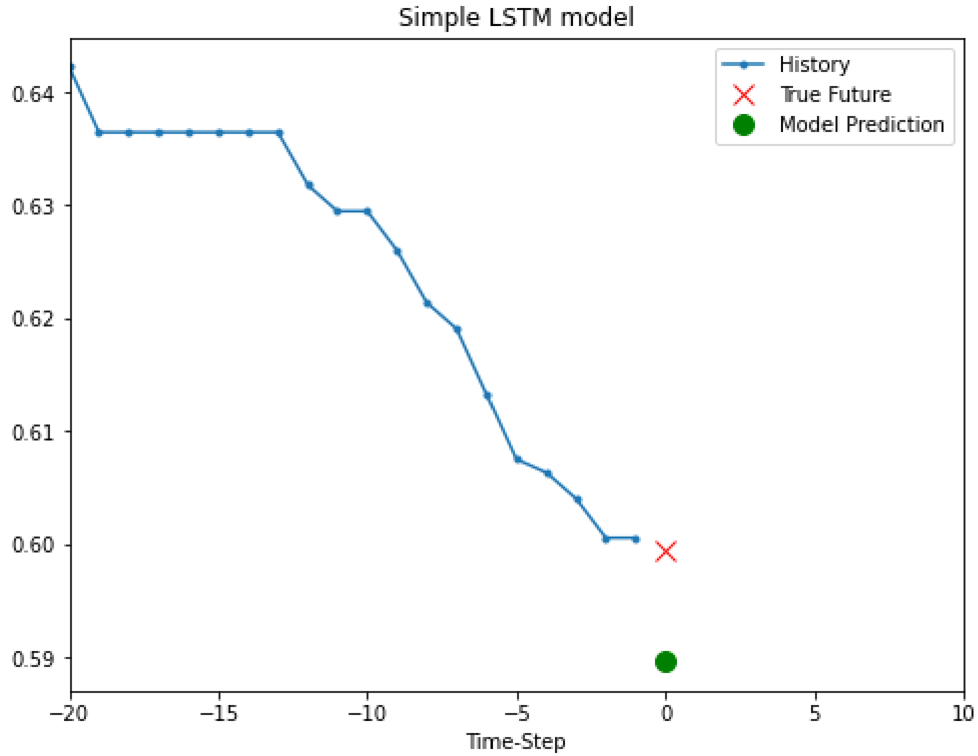
```
200/200 [==============================] - 3s 16ms/step - loss: 0.0263 - val_loss: 0.0197
Epoch 9/10
200/200 [==============================] - 3s 16ms/step - loss: 0.0253 - val_loss: 0.0182
Epoch 10/10
200/200 [==============================] - 3s 16ms/step - loss: 0.0227 - val_loss: 0.0174
```

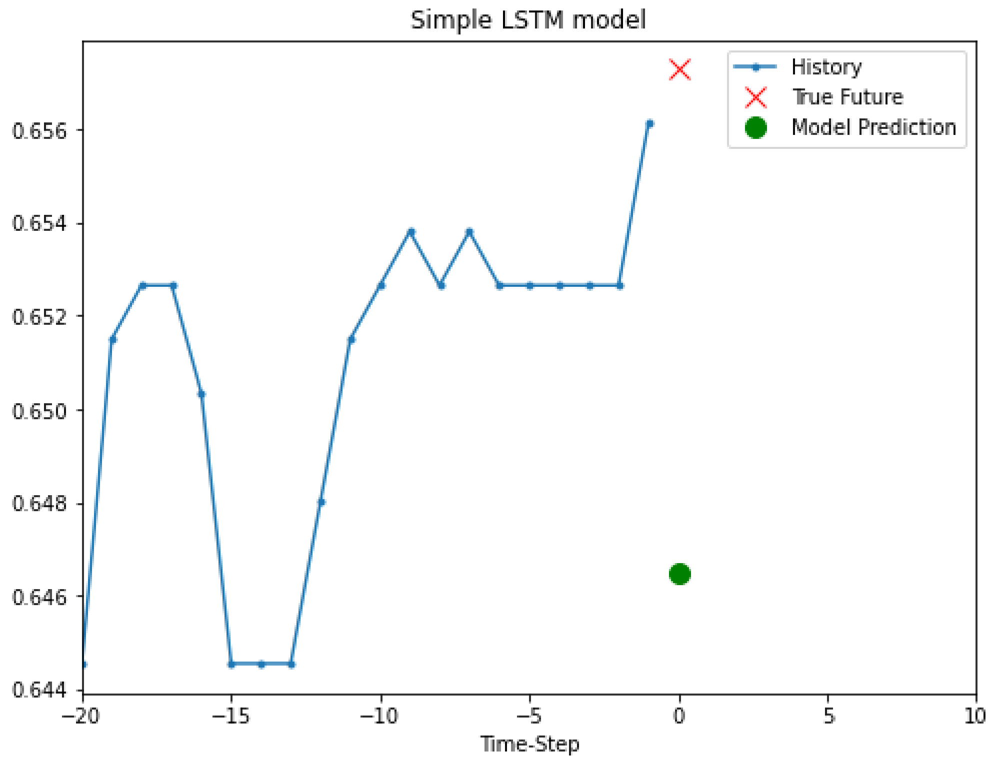
Out[21]:

```
<keras.callbacks.History at 0x13f28f04220>
```

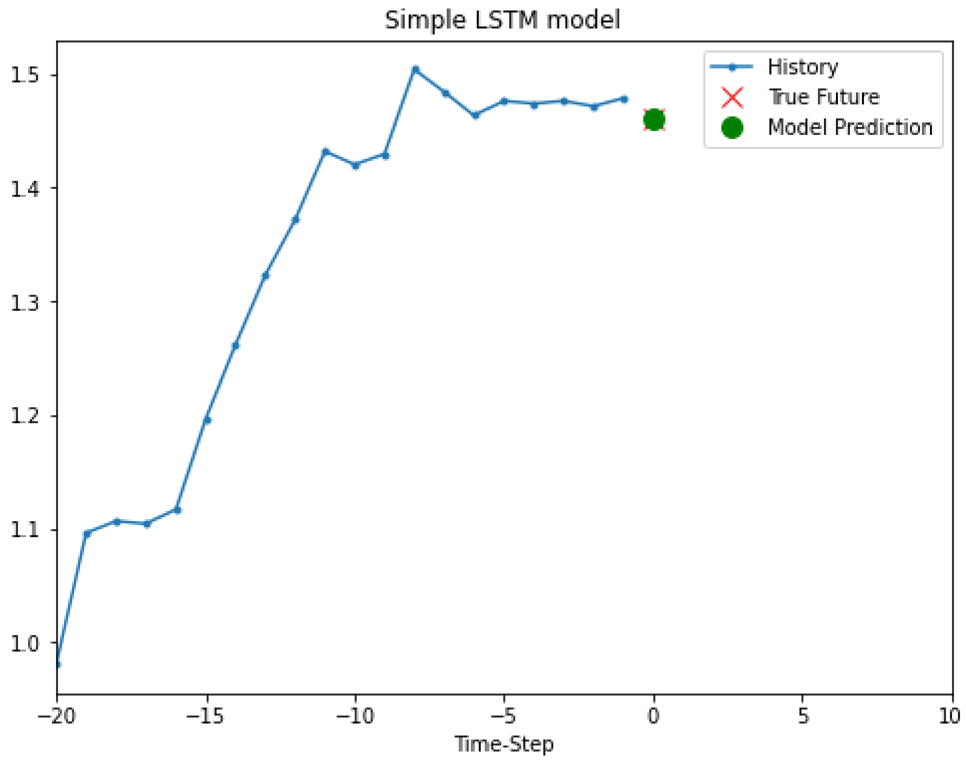
In [22]:

```python
for x, y in val_univariate.take(5): # take 5 random inputs from validation data
  plot = show_plot([x[0].numpy(), y[0].numpy(),
                    simple_lstm_model.predict(x)[0]], 0, 'Simple LSTM model')
  plot.show()
```

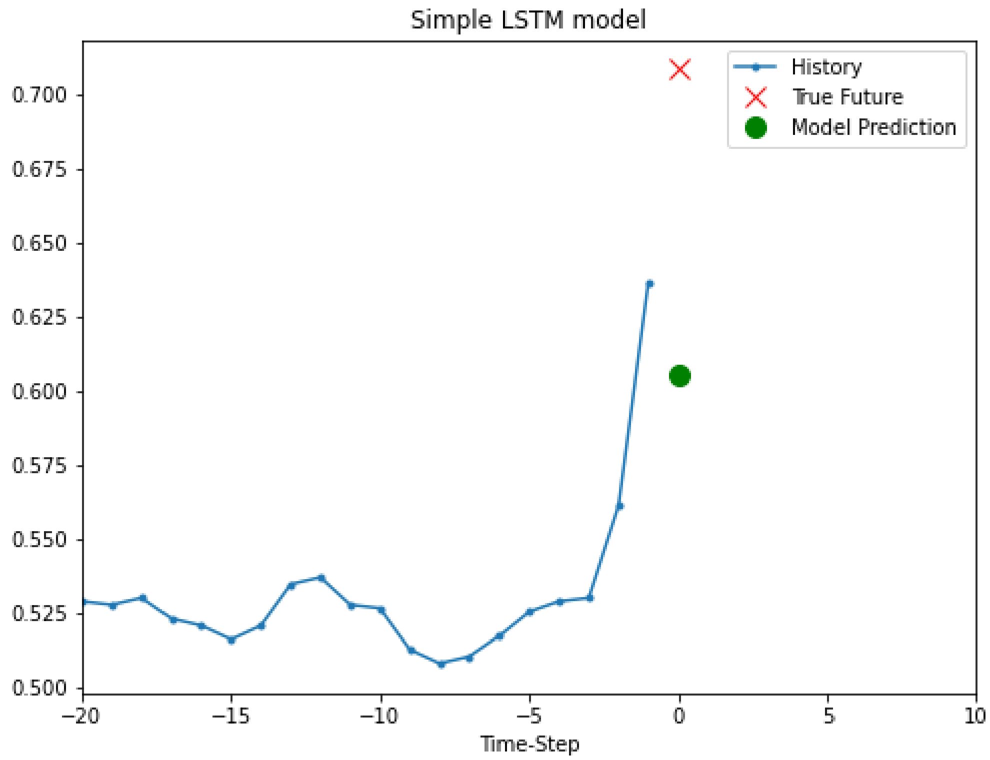



Simple LSTM model
History
True Future
Model Prediction
0.656
0.654
0.652
0.650
0.648
0.646
0.644
−20
−15
−10
−5
0
5
10
Time-Step


Simple LSTM model
History
True Future
Model Prediction
1.60
1.55
1.50
1.45
1.40
1.35
−20
−15
−10
−5
0
5
10
Time-Step



Simple LSTM model
History
True Future
Model Prediction
1.5
1.4
1.3
1.2
1.1
1.0
−20
−15
−10
−5
0
5
10
Time-Step



Multi-variate & single-step forecasting

- Problem definition: Given three features (p, T, rho) at each time stamp in the past, predict the temperature at a single time-stamp in the future.

In [23]:

```
# Features
features_considered = ['p (mbar)', 'T (degC)', 'rho (g/m**3)']

features = df[features_considered]
features.index = df['Date Time']
features.head()
```

Out[23]:

	p (mbar)	T (degC)	rho (g/m**3)
Date Time			
01.01.2009 00:10:00	996.52	-8.02	1307.75

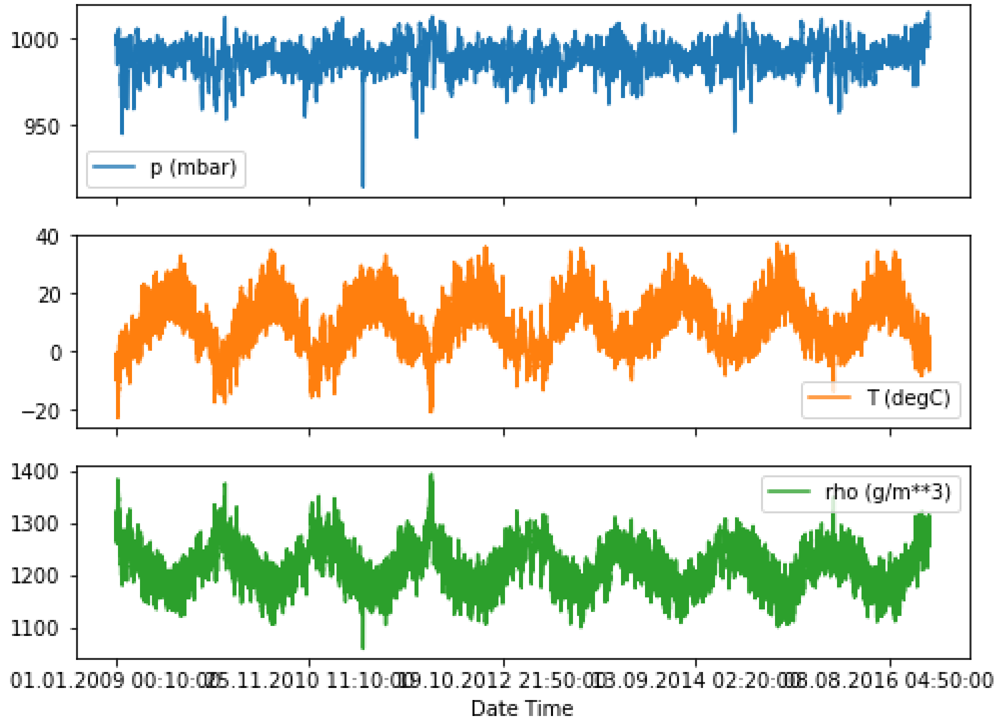
	p (mbar)	T (degC)	rho (g/m**3)
Date Time			
01.01.2009 00:20:00	996.57	-8.41	1309.80
01.01.2009 00:30:00	996.53	-8.51	1310.24
01.01.2009 00:40:00	996.51	-8.31	1309.19
01.01.2009 00:50:00	996.51	-8.27	1309.00

In [24]:

```
features.plot(subplots=True)
```

Out[24]:

```
array([<AxesSubplot:xlabel='Date Time'>, <AxesSubplot:xlabel='Date Time'>,
       <AxesSubplot:xlabel='Date Time'>], dtype=object)
```



In [25]:

```
# Standardize data
```

```python
dataset = features.values
data_mean = dataset[:TRAIN_SPLIT].mean(axis=0)
data_std = dataset[:TRAIN_SPLIT].std(axis=0)

dataset = (dataset-data_mean)/data_std
```

In [26]:

```python
# Same as univariate_data above.

# New params:
# step: instead of taking data for each 10min, do you want to generate data once evrey 6 steps (60min)
# single_step: lables from single timestamp or multiple timesteps


def multivariate_data(dataset, target, start_index, end_index, history_size,
                      target_size, step, single_step=False):
  data = []
  labels = []

  start_index = start_index + history_size
  if end_index is None:
    end_index = len(dataset) - target_size

  for i in range(start_index, end_index):
    indices = range(i-history_size, i, step) # step used here.
    data.append(dataset[indices])

    if single_step: # single_step used here.
      labels.append(target[i+target_size])
    else:
      labels.append(target[i:i+target_size])

  return np.array(data), np.array(labels)
```

In [27]:

```python
# Generate data
past_history = 720 # 720*10 mins
future_target = 72 # 72*10 mins
STEP = 6 # one obs every 6X10min = 60 min => 1 hr

# past history: 7200 mins => 120 hrs, sampling at one sample evry hours
# future_target: 720 mins = > 12 hrs in the future, not next hour
```

```
x_train_single, y_train_single = multivariate_data(dataset, dataset[:, 1], 0,
                                                   TRAIN_SPLIT, past_history,
                                                   future_target, STEP,
                                                   single_step=True)
x_val_single, y_val_single = multivariate_data(dataset, dataset[:, 1],
                                               TRAIN_SPLIT, None, past_history,
                                               future_target, STEP,
                                               single_step=True)

print(x_train_single.shape)
print(y_train_single.shape)
```

```
(299280, 120, 3)
(299280,)
```

In [28]:

```
#TF dataset

train_data_single = tf.data.Dataset.from_tensor_slices((x_train_single, y_train_single))
train_data_single = train_data_single.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()

val_data_single = tf.data.Dataset.from_tensor_slices((x_val_single, y_val_single))
val_data_single = val_data_single.batch(BATCH_SIZE).repeat()

print(train_data_single)
print(val_data_single)
```

```
<RepeatDataset element_spec=(TensorSpec(shape=(None, 120, 3), dtype=tf.float64, name=None), TensorSpec(shape=(None,), dtype=tf.flo
at64, name=None))>
<RepeatDataset element_spec=(TensorSpec(shape=(None, 120, 3), dtype=tf.float64, name=None), TensorSpec(shape=(None,), dtype=tf.flo
at64, name=None))>
```

In [29]:

```
# Model
single_step_model = tf.keras.models.Sequential()
single_step_model.add(tf.keras.layers.LSTM(32,
                                           input_shape=x_train_single.shape[-2:]))
single_step_model.add(tf.keras.layers.Dense(1))

single_step_model.compile(optimizer=tf.keras.optimizers.RMSprop(), loss='mae')

single_step_history = single_step_model.fit(train_data_single, epochs=EPOCHS,
                                            steps_per_epoch=STEPS_PER_EPOCH,
```

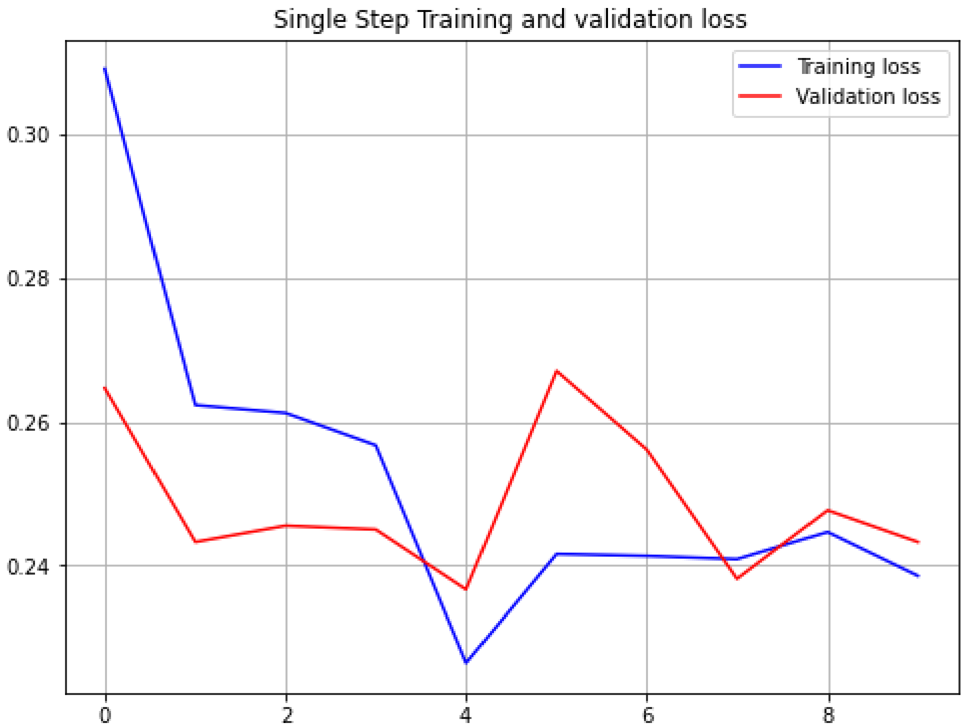
```
                                    validation_data=val_data_single,
                                    validation_steps=50)
```

```
Epoch 1/10
200/200 [==============================] - 48s 225ms/step - loss: 0.3090 - val_loss: 0.2647
Epoch 2/10
200/200 [==============================] - 47s 236ms/step - loss: 0.2623 - val_loss: 0.2433
Epoch 3/10
200/200 [==============================] - 54s 271ms/step - loss: 0.2612 - val_loss: 0.2456
Epoch 4/10
200/200 [==============================] - 51s 255ms/step - loss: 0.2567 - val_loss: 0.2450
Epoch 5/10
200/200 [==============================] - 53s 263ms/step - loss: 0.2265 - val_loss: 0.2367
Epoch 6/10
200/200 [==============================] - 65s 323ms/step - loss: 0.2416 - val_loss: 0.2671
Epoch 7/10
200/200 [==============================] - 64s 320ms/step - loss: 0.2413 - val_loss: 0.2561
Epoch 8/10
200/200 [==============================] - 66s 332ms/step - loss: 0.2409 - val_loss: 0.2382
Epoch 9/10
200/200 [==============================] - 56s 278ms/step - loss: 0.2447 - val_loss: 0.2477
Epoch 10/10
200/200 [==============================] - 62s 308ms/step - loss: 0.2386 - val_loss: 0.2433
```

In [30]:

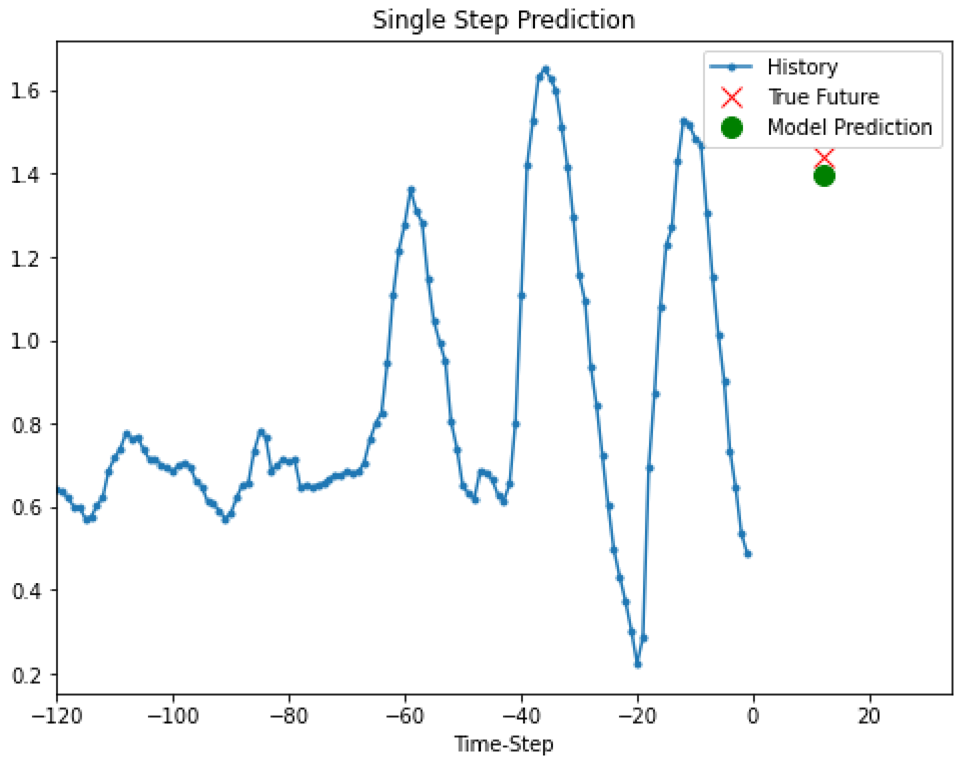
```
# Plot train and validation loss over epochs

def plot_train_history(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.legend()
  plt.grid()

  plt.show()
```

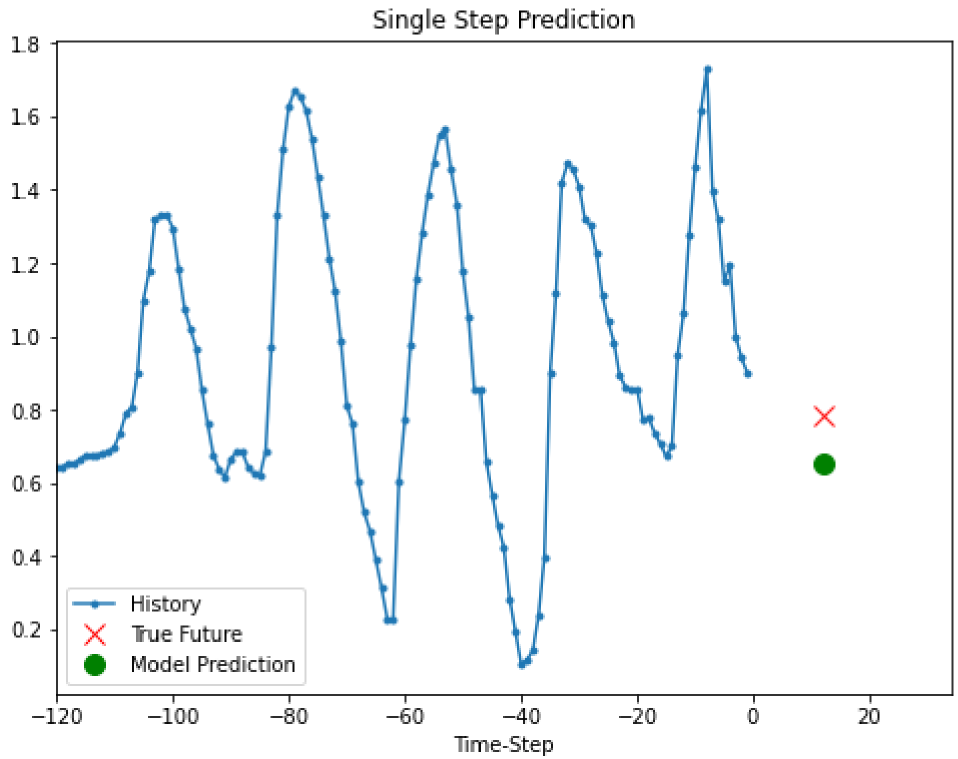
```
plot_train_history(single_step_history,
                   'Single Step Training and validation loss')
```

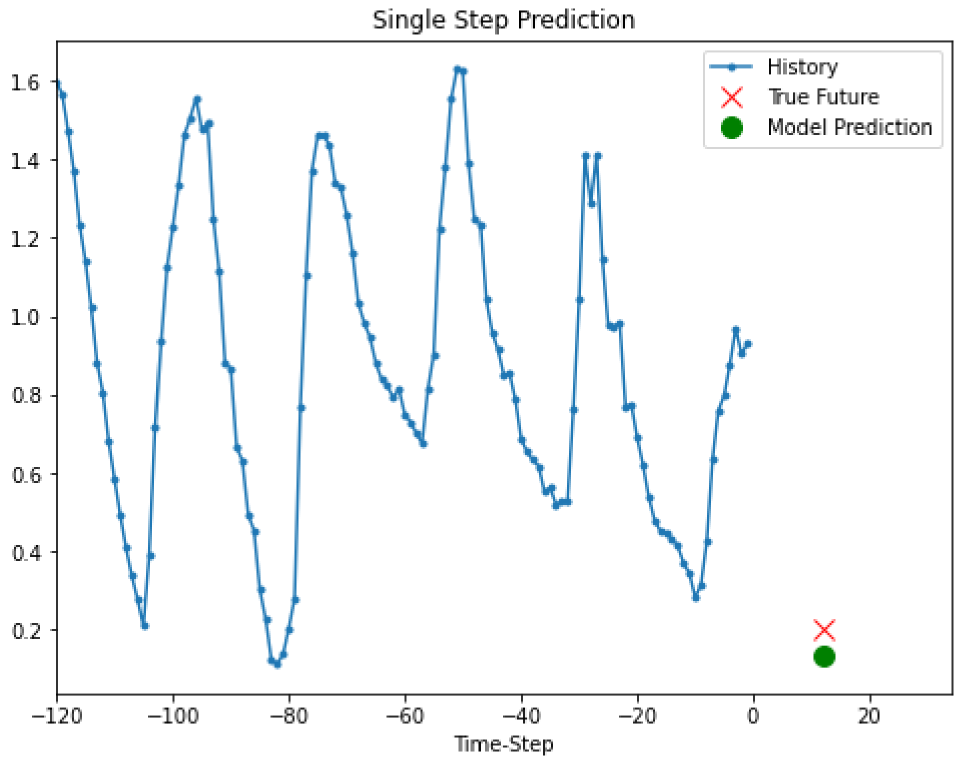


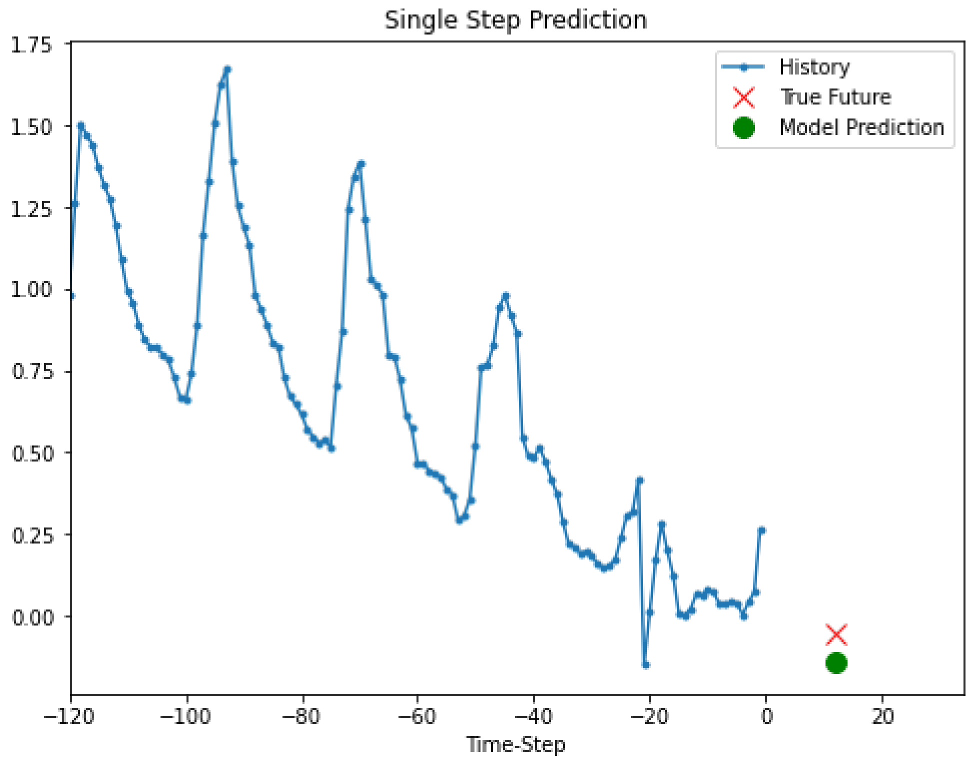
In [31]:

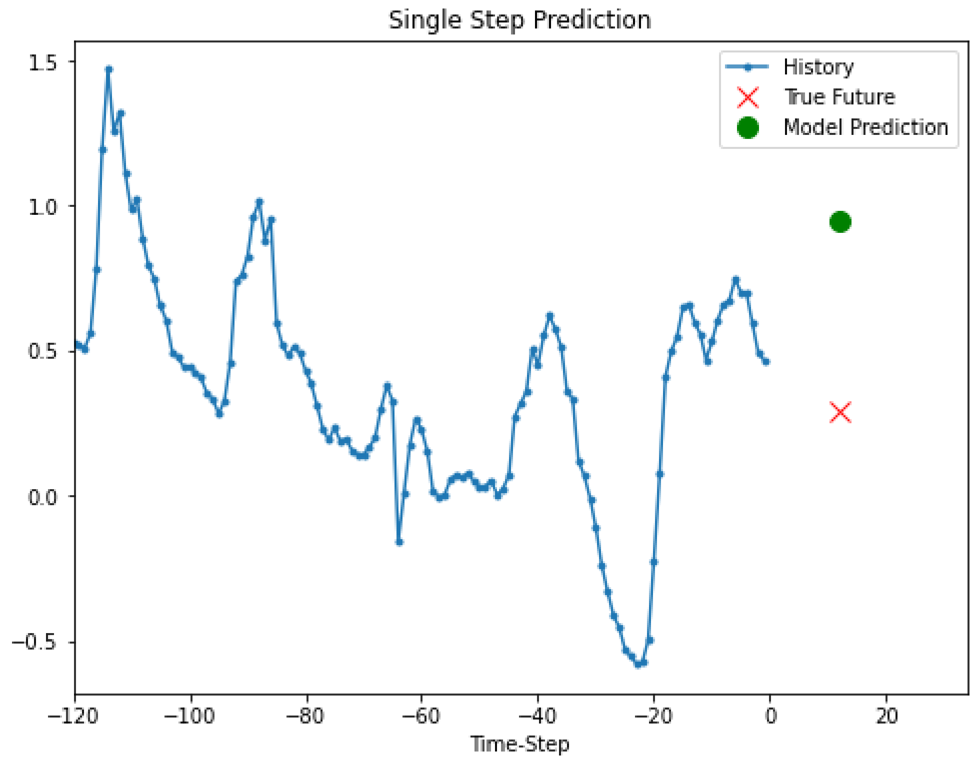
```
# plot time series and predicted values

for x, y in val_data_single.take(5):
  plot = show_plot([x[0][:, 1].numpy(), y[0].numpy(),
                    single_step_model.predict(x)[0]], 12,
                   'Single Step Prediction')
  plot.show()
```


Single Step Prediction
History
True Future
Model Prediction
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
−120
−100
−80
−60
−40
−20
0
20
Time-Step


Single Step Prediction
History
True Future
Model Prediction
Time-Step
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
−120
−100
−80
−60
−40
−20
0
20


Single Step Prediction
History
True Future
Model Prediction
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
−120
−100
−80
−60
−40
−20
0
20
Time-Step


Single Step Prediction
History
True Future
Model Prediction
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−120
−100
−80
−60
−40
−20
0
20
Time-Step



Multi-variate & multi-step forecasting

- Generate multiple future values of temperature

In [32]:

```
# single_step=FALSE default value

future_target = 72 # 72 future values
x_train_multi, y_train_multi = multivariate_data(dataset, dataset[:, 1], 0,
                                                 TRAIN_SPLIT, past_history,
                                                 future_target, STEP)
x_val_multi, y_val_multi = multivariate_data(dataset, dataset[:, 1],
                                             TRAIN_SPLIT, None, past_history,
                                             future_target, STEP)


print(x_train_multi.shape)
print(y_train_multi.shape)
```

```
print(x_val_multi.shape)
print(y_val_multi.shape)
```

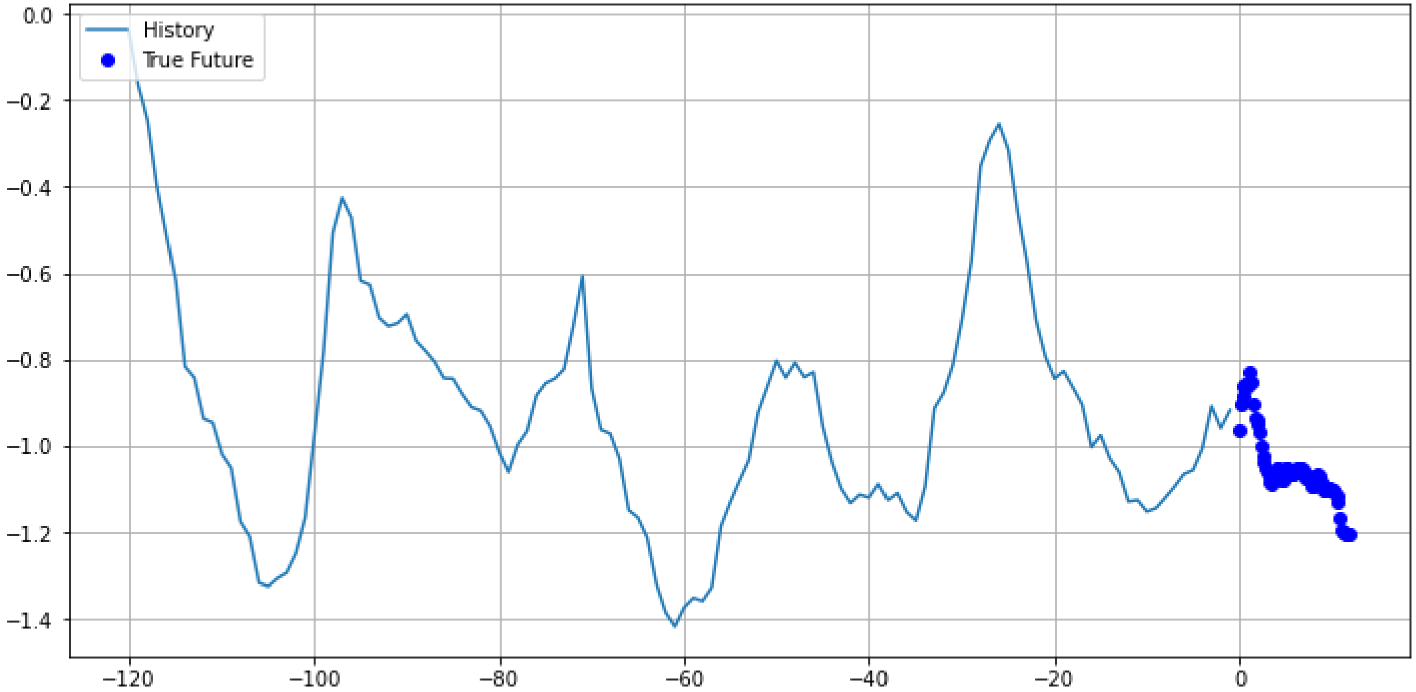
```
(299280, 120, 3)
(299280, 72)
(119759, 120, 3)
(119759, 72)
```

In [33]:

```
# TF DATASET

train_data_multi = tf.data.Dataset.from_tensor_slices((x_train_multi, y_train_multi))
train_data_multi = train_data_multi.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()

val_data_multi = tf.data.Dataset.from_tensor_slices((x_val_multi, y_val_multi))
val_data_multi = val_data_multi.batch(BATCH_SIZE).repeat()
```

In [34]:

```
#plotting function
def multi_step_plot(history, true_future, prediction):
  plt.figure(figsize=(12, 6))
  num_in = create_time_steps(len(history))
  num_out = len(true_future)
  plt.grid()
  plt.plot(num_in, np.array(history[:, 1]), label='History')
  plt.plot(np.arange(num_out)/STEP, np.array(true_future), 'bo',
           label='True Future')
  if prediction.any():
    plt.plot(np.arange(num_out)/STEP, np.array(prediction), 'ro',
             label='Predicted Future')
  plt.legend(loc='upper left')
  plt.show()



for x, y in train_data_multi.take(1):
  multi_step_plot(x[0], y[0], np.array([0]))
```



In [36]:

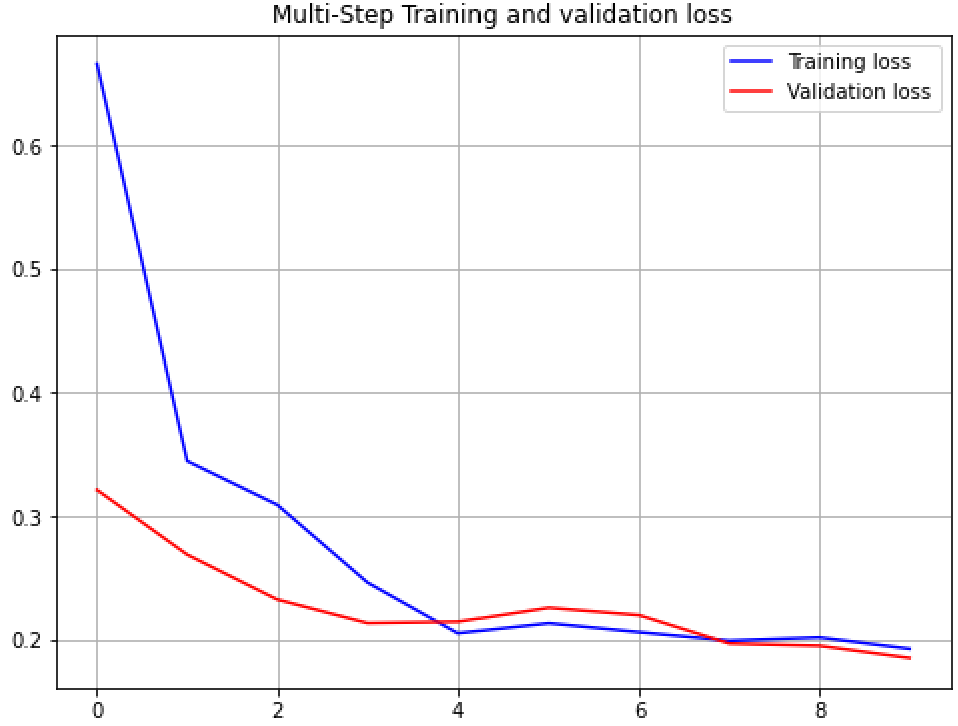
```
multi_step_model = tf.keras.models.Sequential()
multi_step_model.add(tf.keras.layers.LSTM(32,
                                          return_sequences=True,
                                          input_shape=x_train_multi.shape[-2:]))
multi_step_model.add(tf.keras.layers.LSTM(16, activation='relu'))
multi_step_model.add(tf.keras.layers.Dense(72)) # for 72 outputs

multi_step_model.compile(optimizer=tf.keras.optimizers.RMSprop(clipvalue=1.0), loss='mae')

multi_step_history = multi_step_model.fit(train_data_multi, epochs=EPOCHS,
                                          steps_per_epoch=STEPS_PER_EPOCH,
                                          validation_data=val_data_multi,
                                          validation_steps=50)
```

```
Epoch 1/10
200/200 [==============================] - 110s 524ms/step - loss: 0.6657 - val_loss: 0.3212
Epoch 2/10
200/200 [==============================] - 104s 520ms/step - loss: 0.3446 - val_loss: 0.2688
Epoch 3/10
200/200 [==============================] - 107s 535ms/step - loss: 0.3091 - val_loss: 0.2323
```

```
Epoch 4/10
200/200 [==============================] - 106s 529ms/step - loss: 0.2461 - val_loss: 0.2132
Epoch 5/10
200/200 [==============================] - 105s 527ms/step - loss: 0.2050 - val_loss: 0.2144
Epoch 6/10
200/200 [==============================] - 105s 526ms/step - loss: 0.2131 - val_loss: 0.2257
Epoch 7/10
200/200 [==============================] - 106s 532ms/step - loss: 0.2059 - val_loss: 0.2197
Epoch 8/10
200/200 [==============================] - 105s 524ms/step - loss: 0.1990 - val_loss: 0.1967
Epoch 9/10
200/200 [==============================] - 106s 530ms/step - loss: 0.2016 - val_loss: 0.1949
Epoch 10/10
200/200 [==============================] - 110s 551ms/step - loss: 0.1924 - val_loss: 0.1850
```

In [37]:

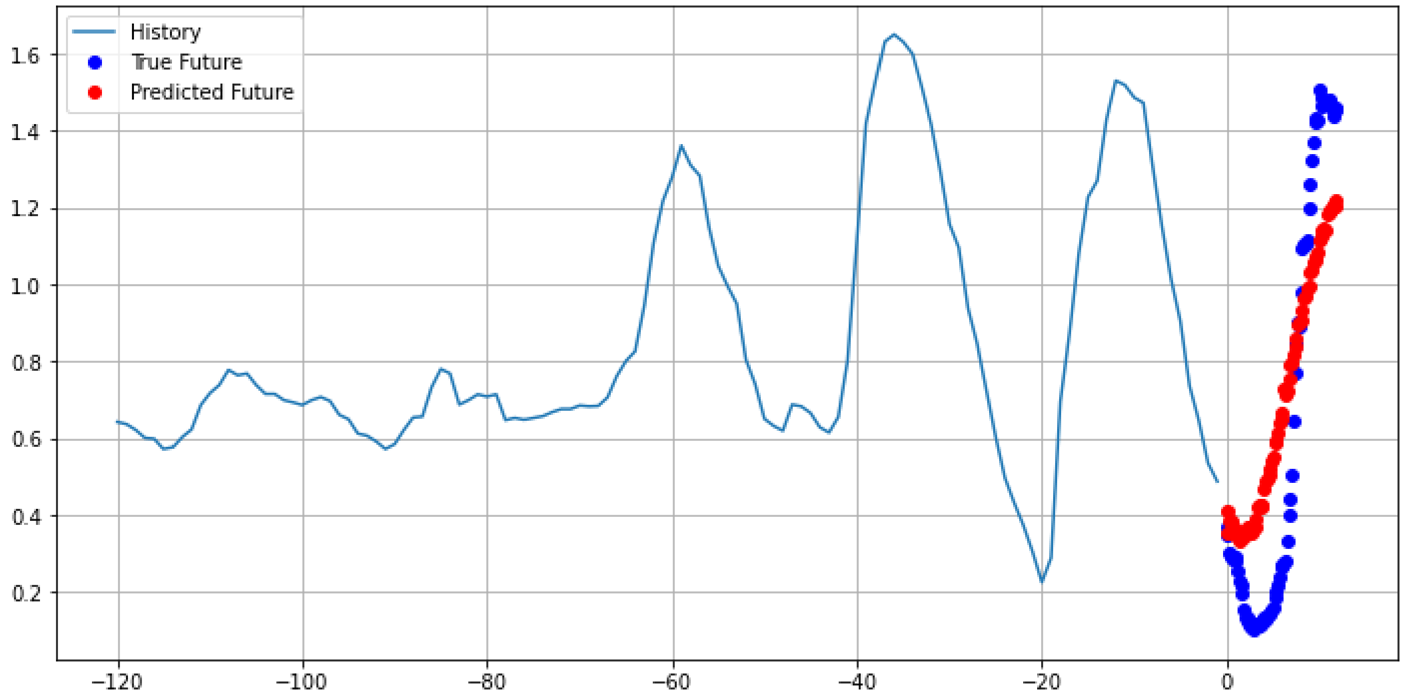
```python
plot_train_history(multi_step_history, 'Multi-Step Training and validation loss')
```

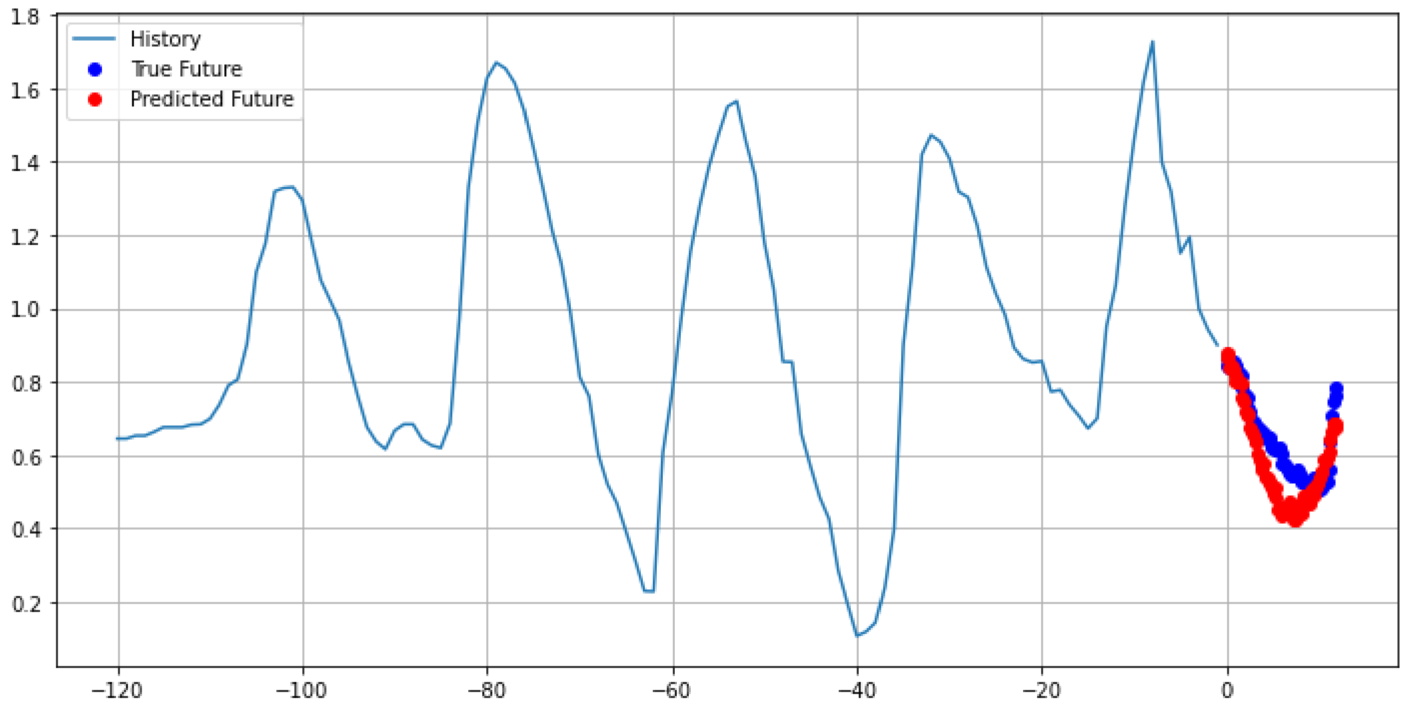


In [38]:

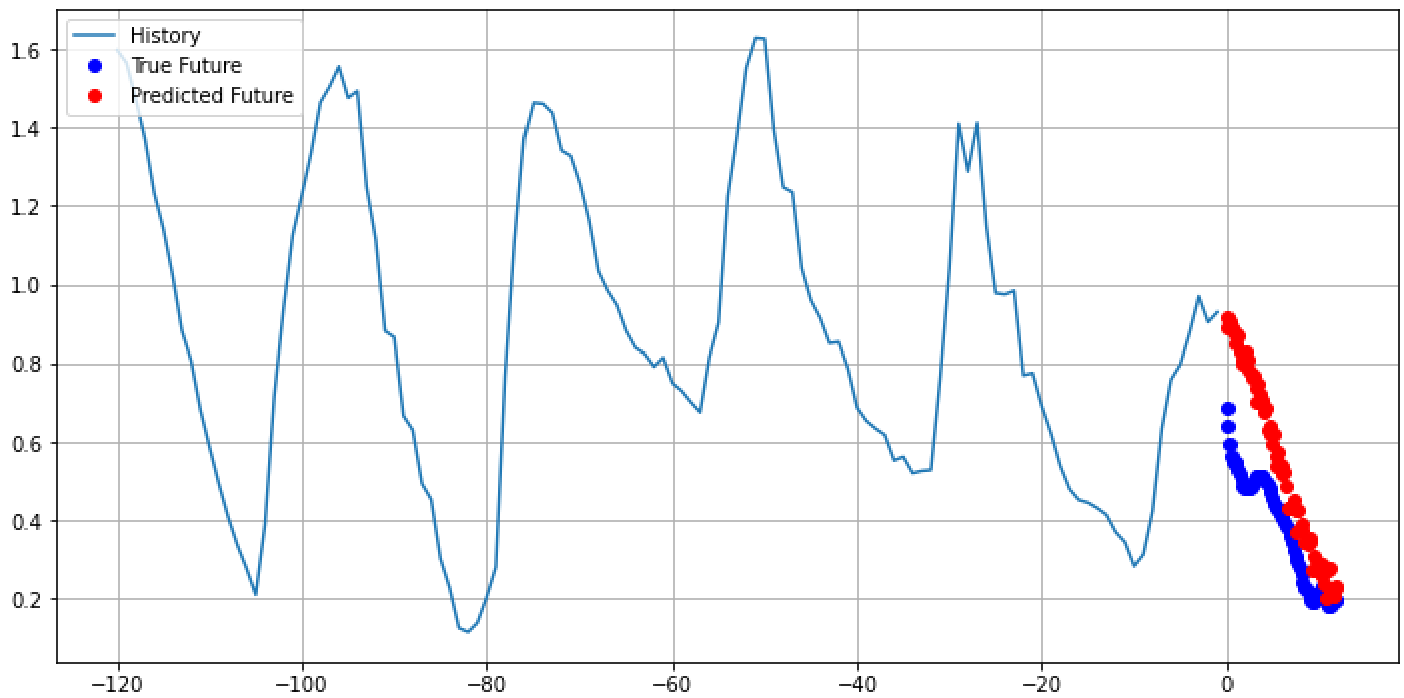
```python
for x, y in val_data_multi.take(3):
```

```
multi_step_plot(x[0], y[0], multi_step_model.predict(x)[0])
```





History
True Future
Predicted Future
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
−120
−100
−80
−60
−40
−20
0


History
True Future
Predicted Future
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
−120
−100
−80
−60
−40
−20
0

In [ ]: