Time series forecasting

This notebook adapts the Tensorflow tutorial on Time series forecasting to data generated from a model for epidemic processes.

Things i changed

-

Steps

1. Imports and setup
2. Load and prepare the generated data
3. Baseline forecasting
4. Univariate LSTM based forecasting
5. Multivariate LSTM based forecasting - Single Step
6. Multivariate LSTM based forecasting - Multiple Steps

Imports and setup

In [6]:
```
import tensorflow as tf

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

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False
```

Load and prepare the generated data

We load data from the ODE model introduced in the notebook "Probability and Information Theory". For each of the 150 virtuel outbreaks (randomized and with different model parameters), we have time series (with 500 steps) for four the variables "Susceptible", "Infected", "Recovered", and "Deceased".

In [7]:
```
csv_path = "./epidemic_process_raw_data.csv"
df = pd.read_csv(csv_path)
df.head()
```

Out[7]:

	1	2	3	4	5	6	7	
0	100.287149	103.541223	95.879814	96.354848	96.980932	97.855310	98.940537	100.1
1	0.993774	1.017558	1.070030	1.116168	1.142078	1.134735	1.182418	1.2
2	0.000000	0.017741	0.036585	0.054735	0.074266	0.096065	0.117691	0.1
3	0.000000	0.000178	0.000364	0.000562	0.000757	0.000947	0.001160	0.0
4	103.489688	100.282780	96.634270	98.532514	99.089272	97.440900	98.416534	101.4

5 rows × 501 columns

In [8]:

```
dfSusceptible = df[df.index % 4 == 0]
dfSusceptible.head()
```

Out[8]:

	1	2	3	4	5	6	7	
0	100.287149	103.541223	95.879814	96.354848	96.980932	97.855310	98.940537	10
4	103.489688	100.282780	96.634270	98.532514	99.089272	97.440900	98.416534	10
8	101.527421	97.711732	96.168179	95.677962	95.575326	96.109792	96.943831	9
12	101.061107	99.112815	106.651686	101.622904	97.726686	95.692173	97.438263	10
16	101.957189	101.898022	100.881113	99.892000	98.939878	98.048565	98.220024	9

5 rows × 501 columns

In [9]:

```
dfInfected = df[df.index % 4 == 1]
dfInfected.head()
```

Out[9]:

	1	2	3	4	5	6	7	8	
1	0.993774	1.017558	1.070030	1.116168	1.142078	1.134735	1.182418	1.272310	1.35
5	1.021677	1.045410	1.120324	1.175914	1.236878	1.306676	1.387931	1.477973	1.54
9	1.020043	1.011238	1.031122	1.048642	1.049479	1.022891	1.035862	1.079177	1.11
13	1.035248	1.014189	1.133178	1.135622	1.157984	1.213088	1.281406	1.359858	1.42
17	1.012666	1.016949	1.053194	1.097599	1.143640	1.192369	1.238880	1.283688	1.32

5 rows × 501 columns

In [10]:

```
dfRecovered = df[df.index % 4 == 2]
dfRecovered.head()
```

Out[10]:

	1	2	3	4	5	6	7	8	9
2	0.0	0.017741	0.036585	0.054735	0.074266	0.096065	0.117691	0.139184	0.163615
6	0.0	0.017909	0.035748	0.056118	0.076620	0.097338	0.119592	0.143024	0.171253
10	0.0	0.016990	0.034644	0.052866	0.071444	0.090609	0.108733	0.126058	0.142408
14	0.0	0.017002	0.036315	0.057484	0.078381	0.098831	0.119563	0.140509	0.166486
18	0.0	0.017589	0.037434	0.056572	0.076275	0.096907	0.116533	0.135387	0.157592

5 rows × 501 columns

In [11]:

```
dfDead = df[df.index % 4 == 3]
dfDead.head()
```

Out[11]:

	1	2	3	4	5	6	7	8	9
3	0.0	0.000178	0.000364	0.000562	0.000757	0.000947	0.001160	0.001389	0.001635
7	0.0	0.000175	0.000351	0.000558	0.000763	0.000968	0.001196	0.001443	0.001719
11	0.0	0.000171	0.000352	0.000538	0.000729	0.000927	0.001126	0.001324	0.001488
15	0.0	0.000181	0.000364	0.000563	0.000774	0.001003	0.001192	0.001351	0.001575
19	0.0	0.000180	0.000358	0.000550	0.000740	0.000931	0.001138	0.001359	0.001574

5 rows × 501 columns

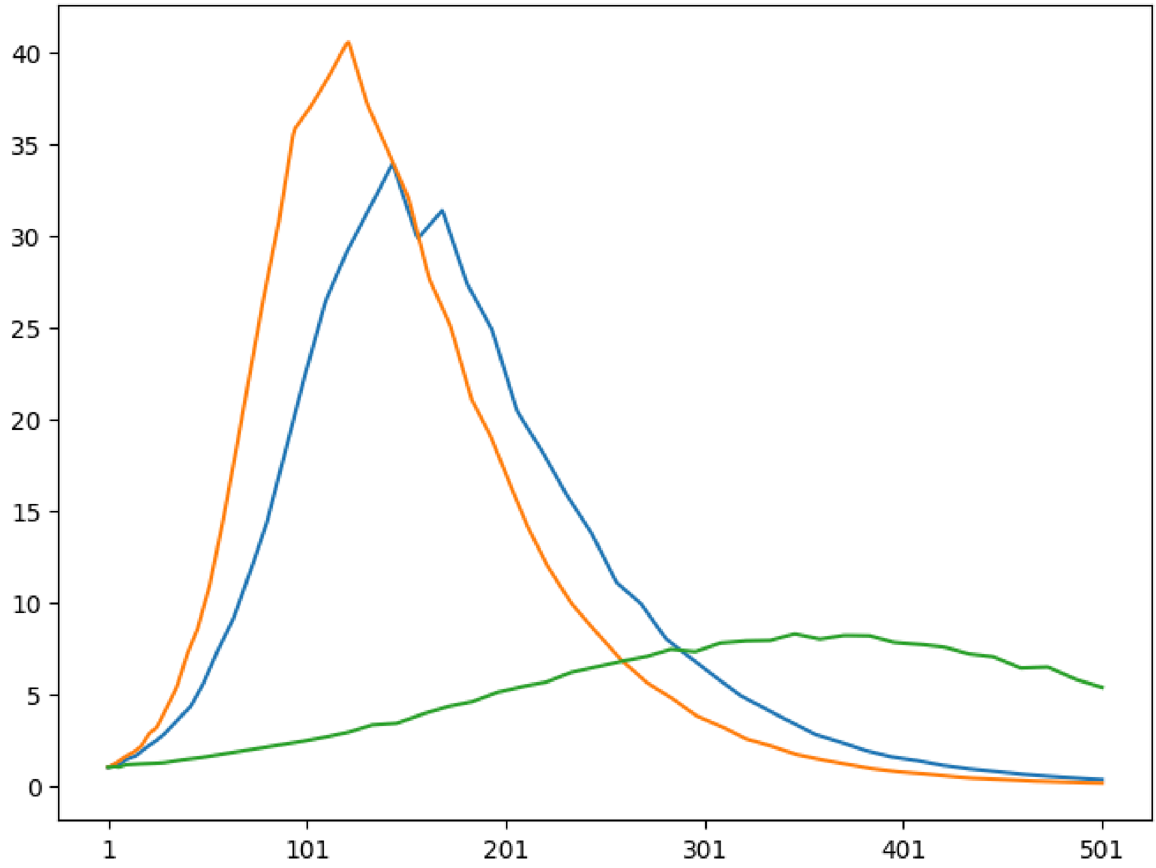
Below a plot of three infection time series for the three first outbreaks.

In [12]:

```
dfInfected.loc[1,:].plot()
dfInfected.loc[5,:].plot()
dfInfected.loc[9,:].plot()
```

Out[12]:

```
<Axes: >
```



We define a 90% / 10% of data for training / testing.

In [13]:
```
dfInfected_arr = dfInfected.values
dfInfected_arr.shape
TRAIN_SPLIT = int(dfInfected_arr.shape[0]-dfInfected_arr.shape[0]*0.1)
TRAIN_SPLIT
```

Out[13]: 135

We standardize the data.

In [14]:
```
uni_train_mean = dfInfected_arr[:TRAIN_SPLIT].mean()
uni_train_std = dfInfected_arr[:TRAIN_SPLIT].std()
uni_data = (dfInfected_arr-uni_train_mean)/uni_train_std
print ('\n Univariate data shape')
print(uni_data.shape)
```

```
 Univariate data shape
(150, 501)
```

We split the data into time series of `univariate_past_history=20` days length and predict the future of the current day, i.e., `univariate_future_target=0`, for the "infected" variable.

In [15]:
```
def univariate_data(dataset, start_series, end_series, history_size, target_size
    data = []
    labels = []
    start_index = history_size
    end_index = len(dataset[0]) - target_size
    for c in range(start_series, end_series):
        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[c][indices], (history_size, 1)))
            labels.append(dataset[c][i+target_size])
    return np.array(data), np.array(labels)
```

In [16]:
```
univariate_past_history = 20 #days
univariate_future_target = 0 #current day

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, len(uni_data),
                                       univariate_past_history,
                                       univariate_future_target)
```

In [17]:
```
print ('Single window of past history')
print (x_train_uni[0])
print ('\n Target number to predict')
print (y_train_uni[0])
print ('\n Number of traing data points')
print (y_train_uni.shape[0])
print ('\n Number of test data points')
print (x_val_uni.shape[0])
```

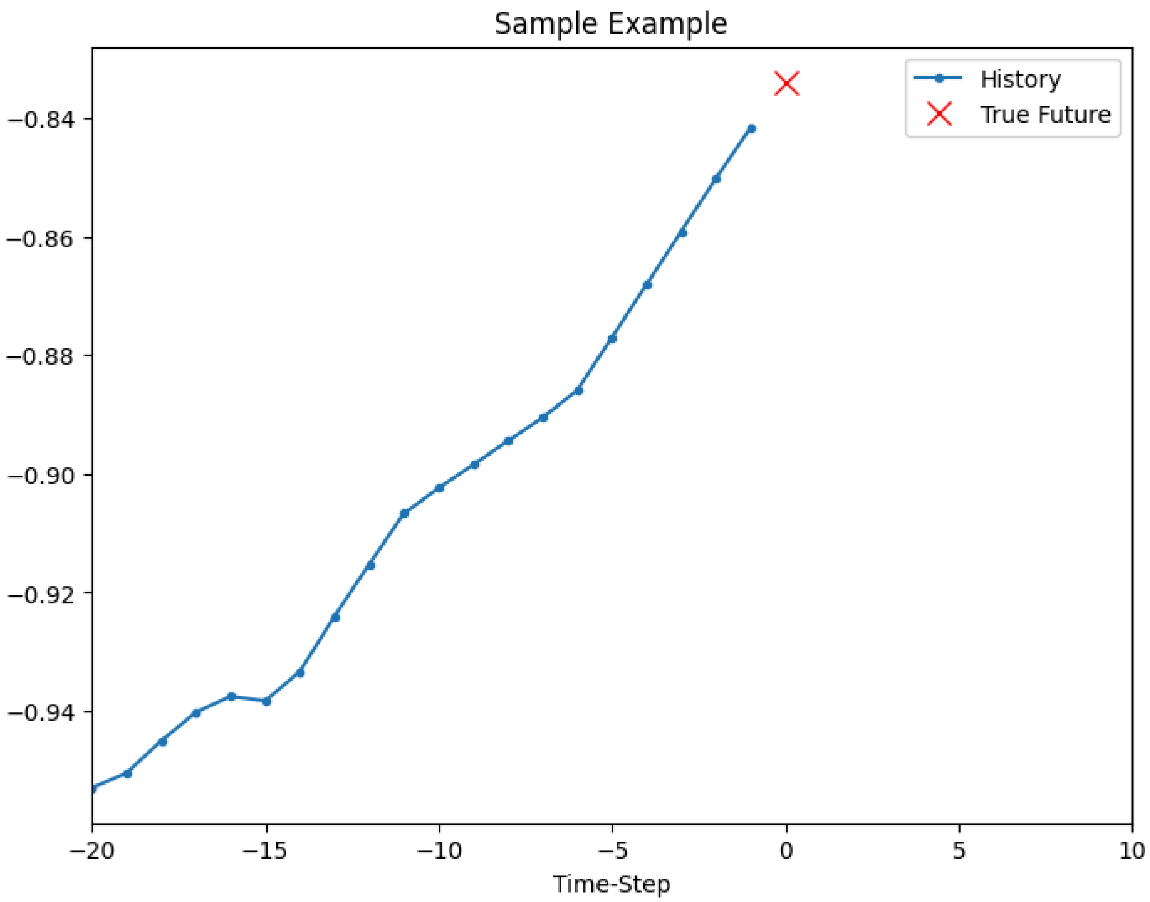
```
Single window of past history
[[-0.95291296]
 [-0.95044298]
 [-0.94499366]
 [-0.9402021 ]
 [-0.93751136]
 [-0.93827393]
 [-0.93332191]
 [-0.92398652]
 [-0.91523643]
 [-0.90667772]
 [-0.90243571]
 [-0.89846308]
 [-0.89449045]
 [-0.89051782]
 [-0.88593997]
 [-0.87701137]
 [-0.86808277]
 [-0.85915417]
 [-0.85022557]
 [-0.84167481]]

 Target number to predict
-0.8339932964893617

 Number of traing data points
64935

 Number of test data points
7215
```

In [18]:
```
def create_time_steps(length):
    return list(range(-length, 0))
```

In [19]:
```python
def show_plot(plot_data, delta, title):
    labels = ['History', 'True Future', 'Model Prediction']
    marker = ['.-', 'rx', 'go']
    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
        else:
            plt.plot(time_steps, plot_data[i].flatten(), marker[i], label=labels
    plt.legend()
    plt.xlim([time_steps[0], (future+5)*2])
    plt.xlabel('Time-Step')
    return plt
```

In [20]:
```python
show_plot([x_train_uni[0], y_train_uni[0]], 0, 'Sample Example')
```

Out[20]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\kemal\\AppData\\Local\\Packages\\P
ythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\\LocalCache\\local-packages
\\Python310\\site-packages\\matplotlib\\pyplot.py'>
```



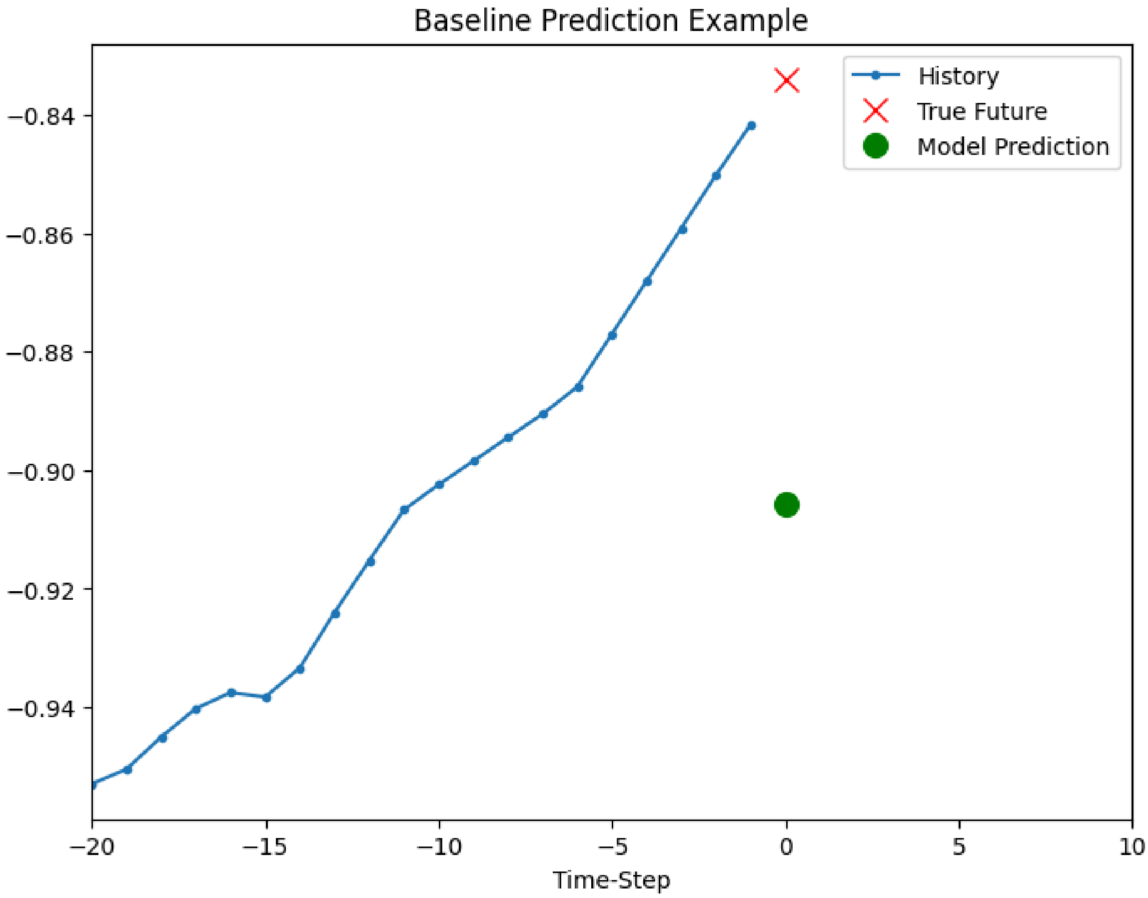
Baseline forecasting

Predicts the mean of the `history`.

In [21]:
```
def baseline(history):
    return np.mean(history)
```

In [22]:
```
show_plot([x_train_uni[0], y_train_uni[0], baseline(x_train_uni[0])], 0, 'Baseli
```

Out[22]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\kemal\\AppData\\Local\\Packages\\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\\LocalCache\\local-packages\\Python310\\site-packages\\matplotlib\\pyplot.py'>
```



Univariate LSTM based forecasting

In [23]:
```
print (x_train_uni.shape)
print (y_train_uni.shape)
x_train_uni.dtype
```

```
(64935, 20, 1)
(64935,)
```

Out[23]:
```
dtype('float64')
```

Batching and resampling; the dataset is repeated indefinitely. Check the tutorial for the details.

In [24]:
```
BATCH_SIZE = 256
BUFFER_SIZE = 10000

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

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

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

train_univariate
```

Out[24]:
```
<_RepeatDataset element_spec=(TensorSpec(shape=(None, 20, 1), dtype=tf.float64,
name=None), TensorSpec(shape=(None,), dtype=tf.float64, name=None))>
```

We define the first LSTM model with 8 units.

In [25]:
```
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')
simple_lstm_model.summary()
x_train_uni.shape[-2:]
```

```
C:\Users\kemal\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2
kfra8p0\LocalCache\local-packages\Python310\site-packages\keras\src\layers\rnn\rn
n.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a lay
er. When using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 8)	320
dense (Dense)	(None, 1)	9

Total params: 329 (1.29 KB)

Trainable params: 329 (1.29 KB)

Non-trainable params: 0 (0.00 B)

Out[25]: (20, 1)

In [26]:
```
for x, y in val_univariate.take(1):
    print(simple_lstm_model.predict(x).shape)
    print(y.shape)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
(256, 1)
(256,)
```

When passing an indefinitely repeated training data set, we need to specify the numbre of steps per training interval (epoch).

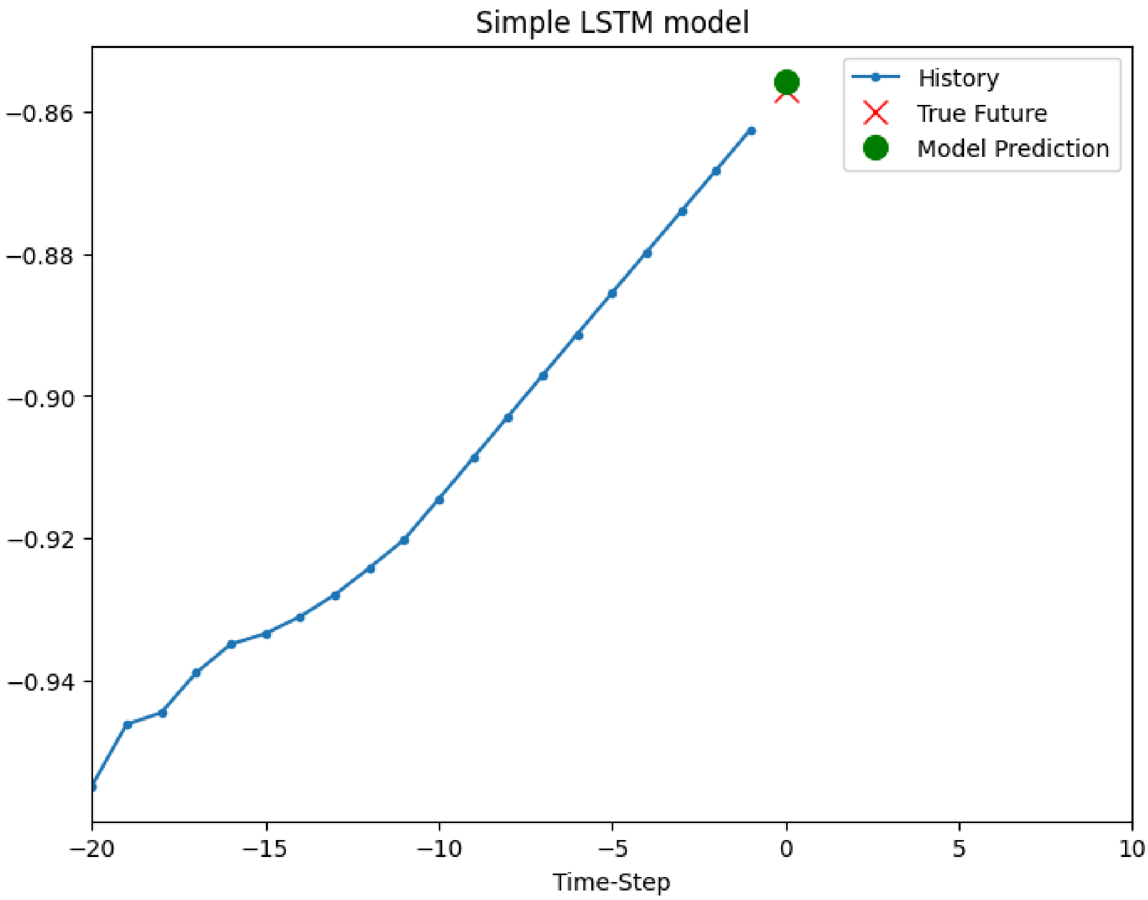
In [27]:
```
EVALUATION_INTERVAL = 2000
EPOCHS = 10

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

```
Epoch 1/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 18s 8ms/step - loss: 0.1180 - val_loss: 0.0034
Epoch 2/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - loss: 0.0036 - val_loss: 0.0021
Epoch 3/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - loss: 0.0025 - val_loss: 0.0017
Epoch 4/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 14s 7ms/step - loss: 0.0022 - val_loss: 0.0019
Epoch 5/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0020 - val_loss: 0.0018
Epoch 6/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0019 - val_loss: 0.0022
Epoch 7/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0019 - val_loss: 0.0021
Epoch 8/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0017 - val_loss: 0.0011
Epoch 9/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - loss: 0.0016 - val_loss: 0.0014
Epoch 10/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0016 - val_loss: 0.0013
```

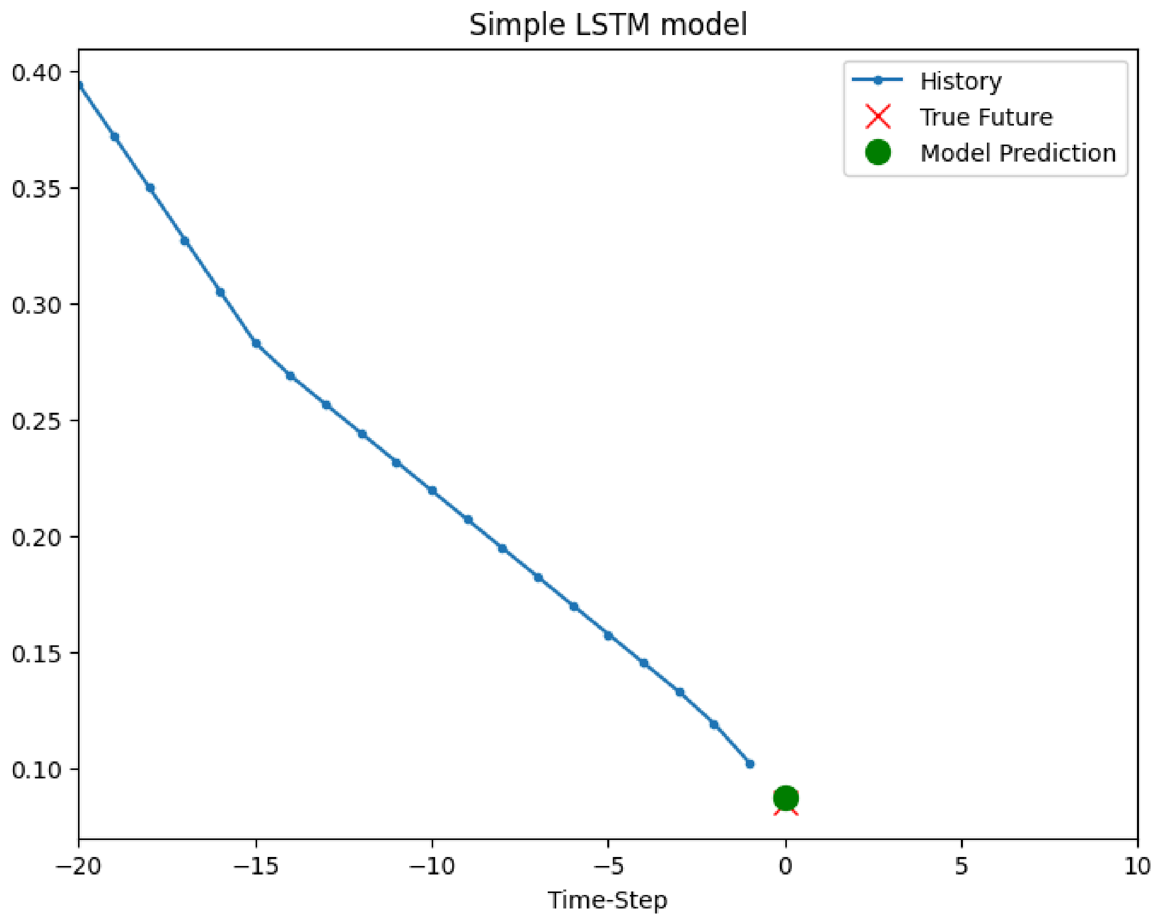
Out[27]: <keras.src.callbacks.history.History at 0x1a409c7d8a0>

In [28]:
```
for x, y in val_univariate.take(3):
    plot = show_plot([x[0].numpy(), y[0].numpy(), simple_lstm_model.predict(x)[0
    plot.show()
```

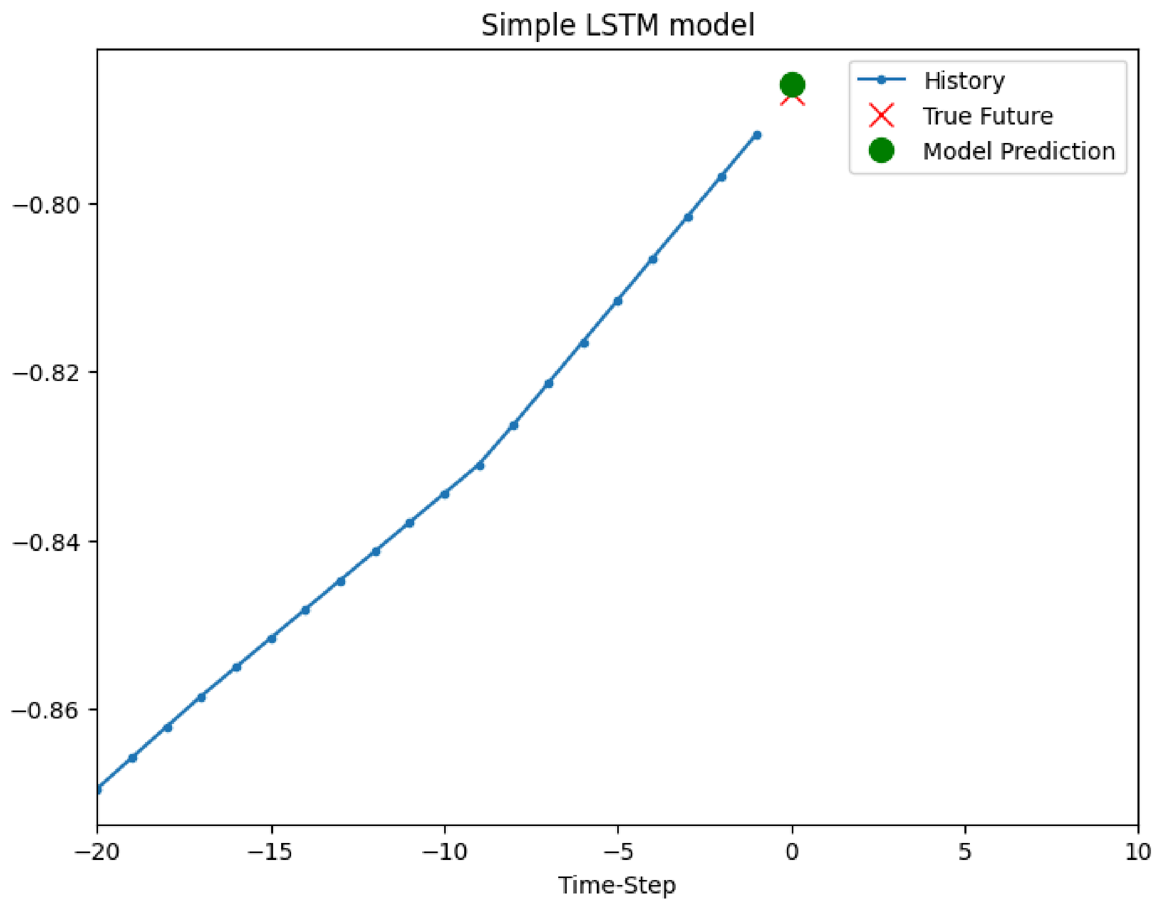
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```




```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```



8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step

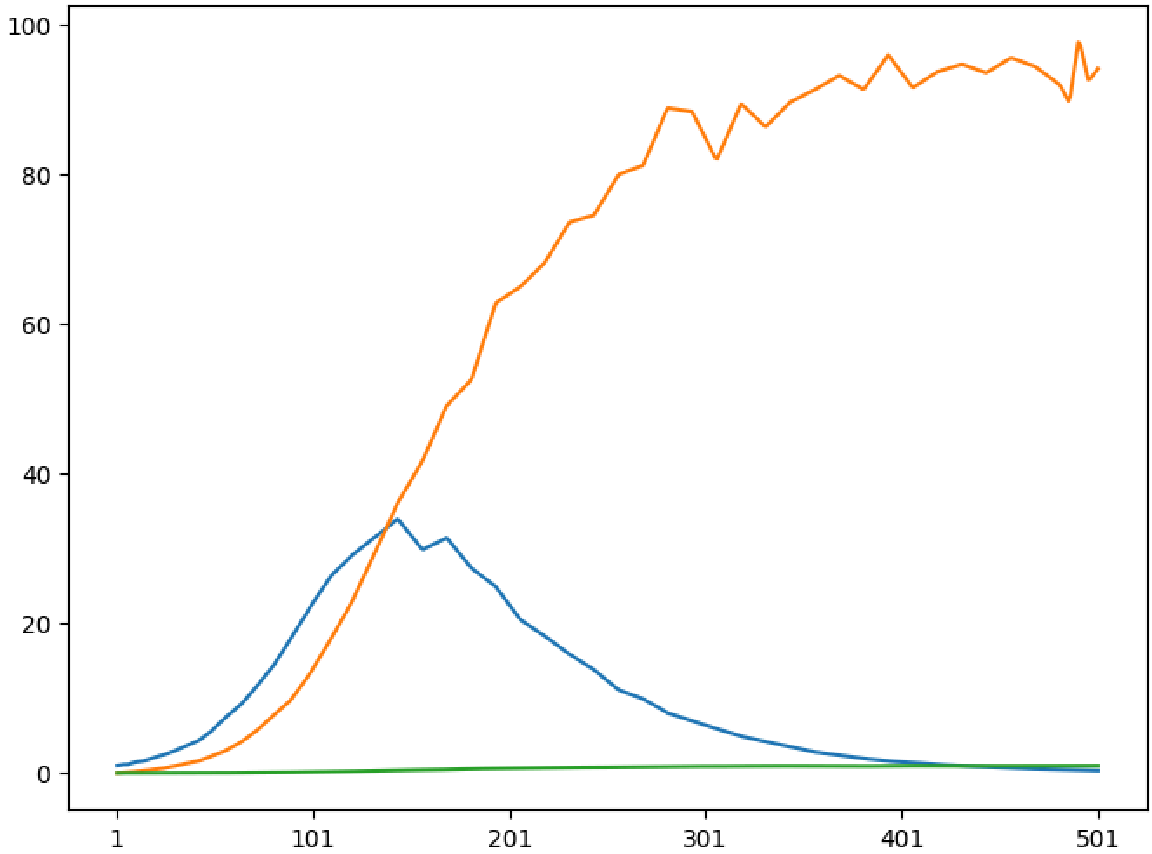


Multivariate LSTM based forecasting - Single Step

We use three variables "Infected", "Recovered", and "Deceased", to forcast "Infected" at one single day in the future.

Here a plot of the time series of the three variables for one outbreak.

In [29]:
```
dfInfected.loc[1,:].plot()
dfRecovered.loc[2,:].plot()
dfDead.loc[3,:].plot()
dfInfected = dfInfected.values
dfRecovered_arr = dfRecovered.values
dfDead_arr = dfDead.values
```



We prepare the dataset.

In [30]:
```
#as before
dfInfected_train_mean = dfInfected_arr[:TRAIN_SPLIT].mean()
dfInfected_train_std = dfInfected_arr[:TRAIN_SPLIT].std()
dfInfected_data = (dfInfected_arr-dfInfected_train_mean)/dfInfected_train_std
#for Recovered
dfRecovered_train_mean = dfRecovered_arr[:TRAIN_SPLIT].mean()
dfRecovered_train_std = dfRecovered_arr[:TRAIN_SPLIT].std()
dfRecovered_data = (dfRecovered_arr-dfRecovered_train_mean)/dfRecovered_train_st
#for Dead
dfDead_train_mean = dfDead_arr[:TRAIN_SPLIT].mean()
dfDead_train_std = dfDead_arr[:TRAIN_SPLIT].std()
dfDead_data = (dfDead_arr-dfDead_train_mean)/dfDead_train_std
```

In [31]:
```
dataset = np.array([dfInfected_data, dfRecovered_data, dfDead_data])
dataset.shape
print ('\n Multivariate data shape')
print(dataset.shape)
```

```
Multivariate data shape
(3, 150, 501)
```

In [32]:
```
def multivariate_data(dataset, target, start_series, end_series, history_size,
                      target_size, step, single_step=False):
    data = []
    labels = []
    start_index = history_size
    end_index = len(dataset[0][0]) - target_size
    for c in range(start_series, end_series):
        for i in range(start_index, end_index):
            indices = range(i-history_size, i, step)
            one = dataset[0][c][indices]
            two = dataset[1][c][indices]
            three = dataset[2][c][indices]
            data.append(np.transpose(np.array([one, two, three])))

            if single_step:
                labels.append(target[c][i+target_size])
            else:
                labels.append(np.transpose(target[c][i:i+target_size]))
    return np.array(data), np.array(labels)
```

We get training and valdation data for time series with a `past_history = 20` days for every other day (`STEP = 2`) and want to predict the "Infected" five days ahead (`future_target = 5`).

In [33]:
```
past_history = 20
future_target = 5
STEP = 2

x_train_single, y_train_single = multivariate_data(dataset, dfInfected_data, 0,
                                                   past_history, future_target,
                                                   single_step=True)
x_val_single, y_val_single = multivariate_data(dataset, dfInfected_data, TRAIN_S
                                               past_history, future_target, STEP
                                               single_step=True)
```

In [34]:
```
print ('Single window of past history : {}'.format(x_train_single[0].shape))
print(dataset.shape)
```

```
Single window of past history : (10, 3)
(3, 150, 501)
```

As before, batching and resampling; the dataset is repeated indefinitely.

In [35]:
```
train_data_single = tf.data.Dataset.from_tensor_slices((x_train_single, y_train_
train_data_single = train_data_single.cache().shuffle(BUFFER_SIZE).batch(BATCH_S

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()
```

In [36]:
```
single_step_model = tf.keras.models.Sequential()
single_step_model.add(tf.keras.layers.LSTM(32, input_shape=x_train_single.shape[
single_step_model.add(tf.keras.layers.Dense(1))

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

```
single_step_model.summary()
x_train_single.shape[-2:]
```

```
C:\Users\kemal\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\keras\src\layers\rnn\rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 32)	4,608
dense_1 (Dense)	(None, 1)	33

Total params: 4,641 (18.13 KB)

Trainable params: 4,641 (18.13 KB)

Non-trainable params: 0 (0.00 B)

Out[36]: (10, 3)

In [37]:
```
for x, y in val_data_single.take(1):
    print(single_step_model.predict(x).shape)
print ('\n Number of traing data points')
print (x_train_single.shape[0])
print ('\n Number of test data points')
print (x_val_single.shape[0])
```

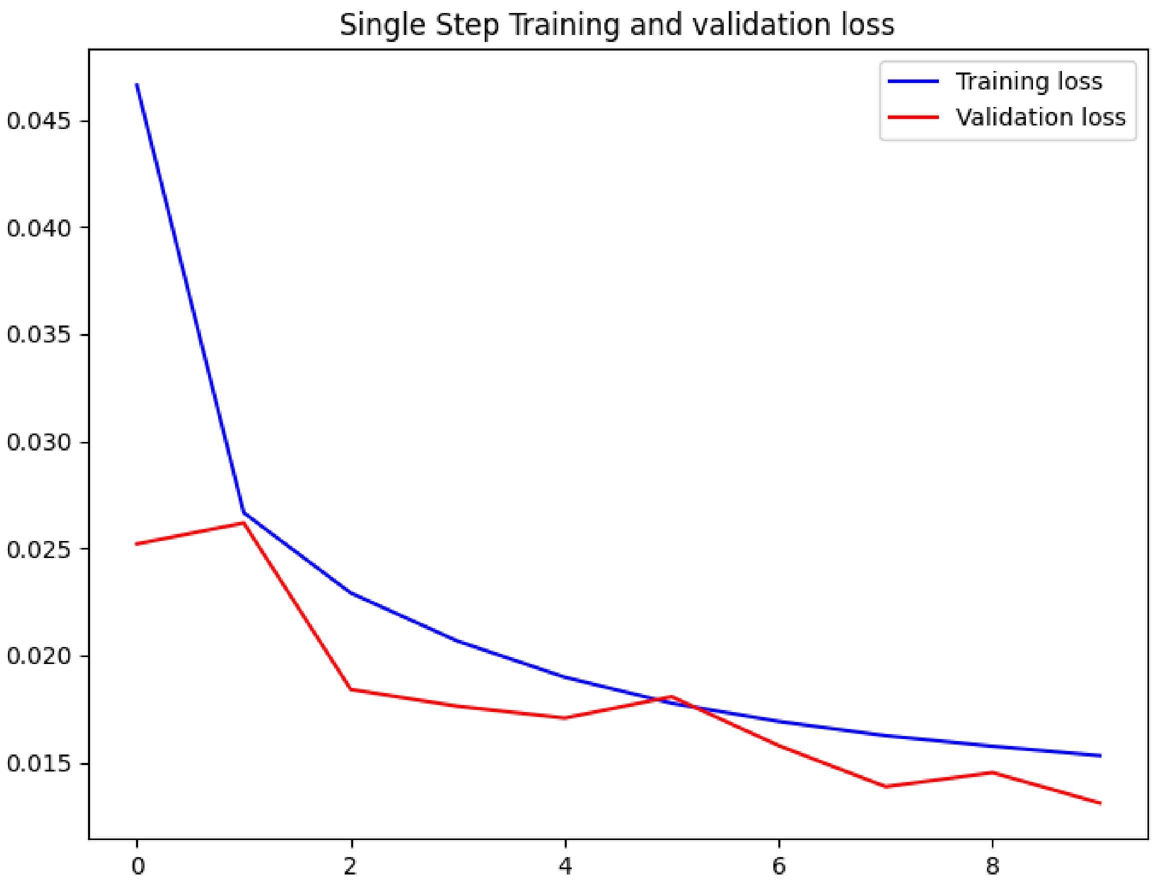
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
(256, 1)

 Number of traing data points
64260

 Number of test data points
7140
```

In [38]:
```
single_step_history = single_step_model.fit(train_data_single, epochs=EPOCHS,
                                            steps_per_epoch=EVALUATION_INTERVAL,
                                            validation_data=val_data_single,
                                            validation_steps=50)
```

```
Epoch 1/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 17s 8ms/step - loss: 0.0891 - val_loss: 0.0252
Epoch 2/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0282 - val_loss: 0.0262
Epoch 3/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0237 - val_loss: 0.0184
Epoch 4/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 17s 8ms/step - loss: 0.0211 - val_loss: 0.0176
Epoch 5/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0193 - val_loss: 0.0171
Epoch 6/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 17s 8ms/step - loss: 0.0180 - val_loss: 0.0181
Epoch 7/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0171 - val_loss: 0.0158
Epoch 8/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 15s 7ms/step - loss: 0.0164 - val_loss: 0.0139
Epoch 9/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0159 - val_loss: 0.0145
Epoch 10/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - loss: 0.0154 - val_loss: 0.0131
```

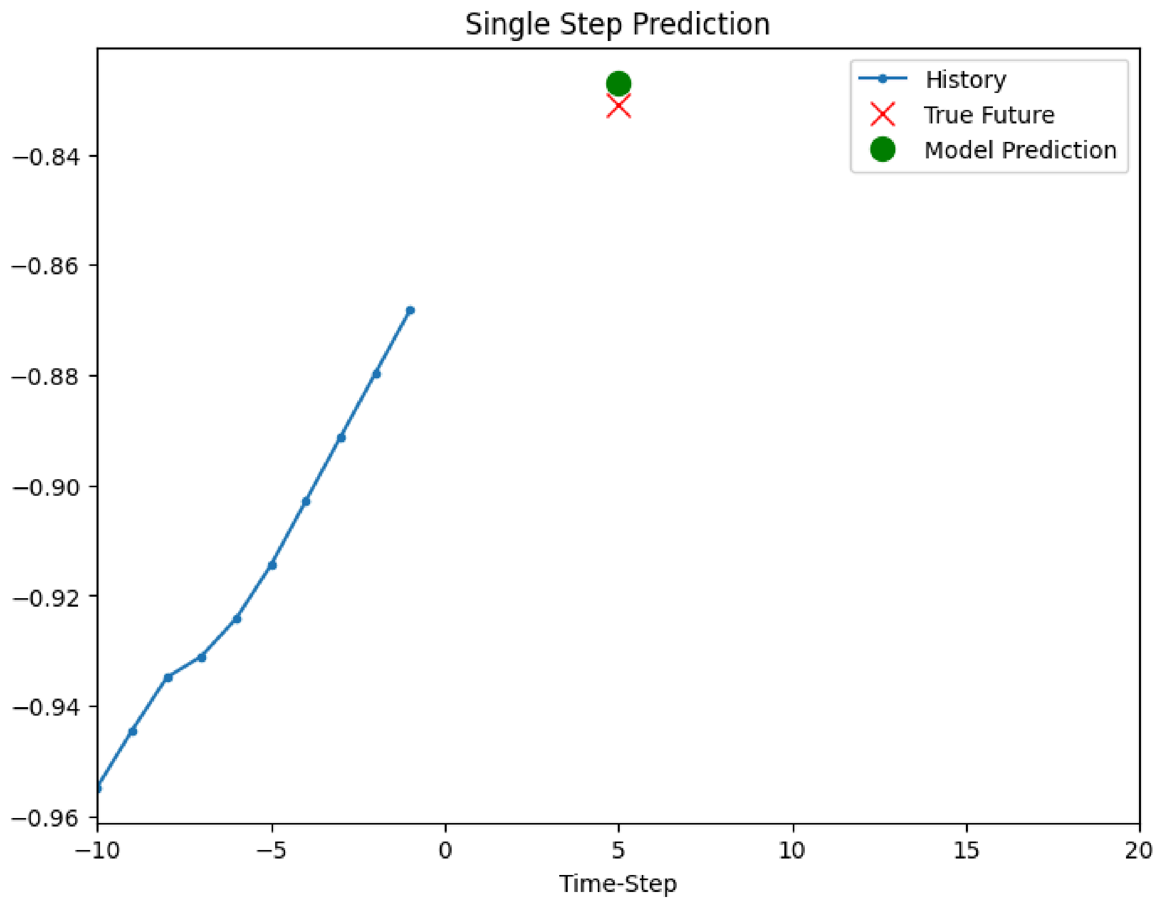
In [39]:
```python
def plot_train_history(history, title):
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title(title)
    plt.legend()
    plt.show()
```

In [40]:
```python
plot_train_history(single_step_history,'Single Step Training and validation loss
```

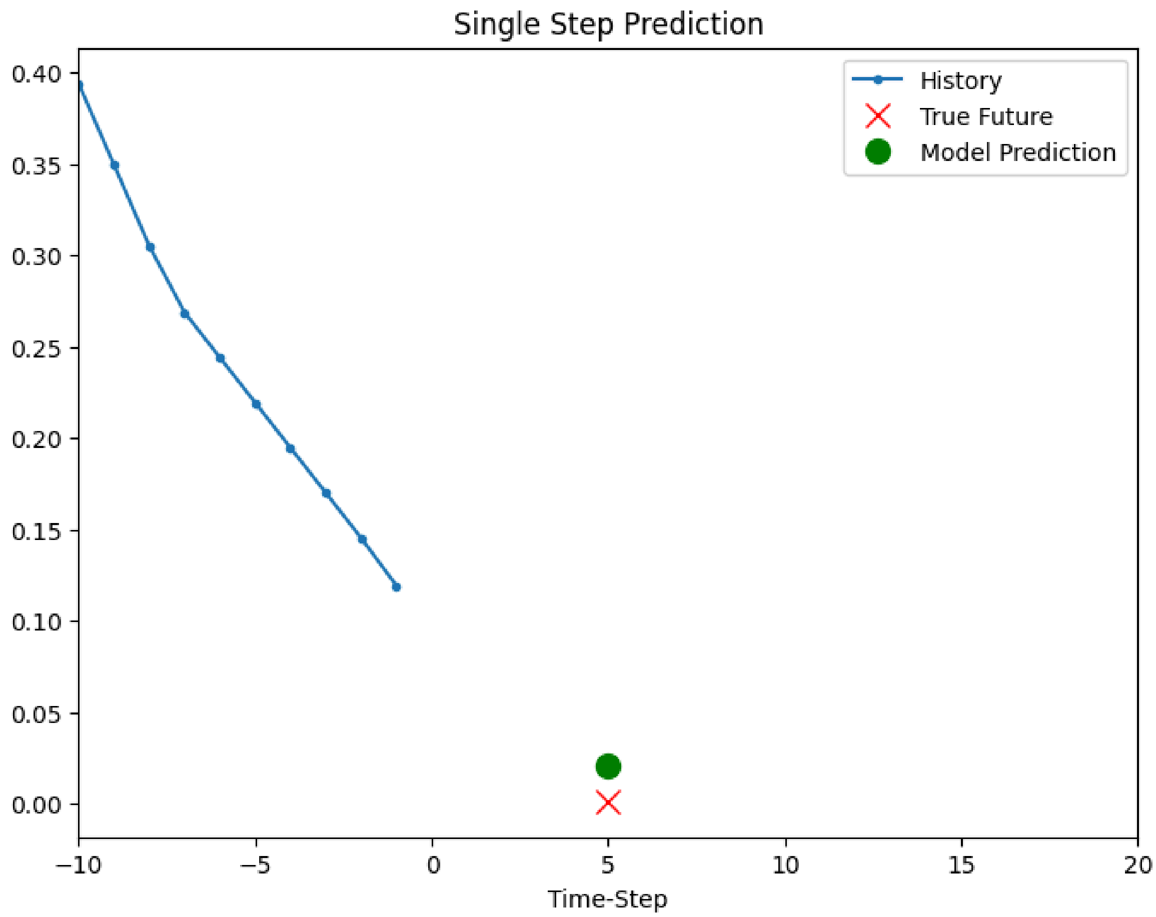


In [41]:
```
for x, y in val_data_single.take(3):
    plot = show_plot([x[0][:, 0].numpy(), y[0].numpy(),
                      single_step_model.predict(x)[0]], future_target,
                     'Single Step Prediction')
    plot.show()
```

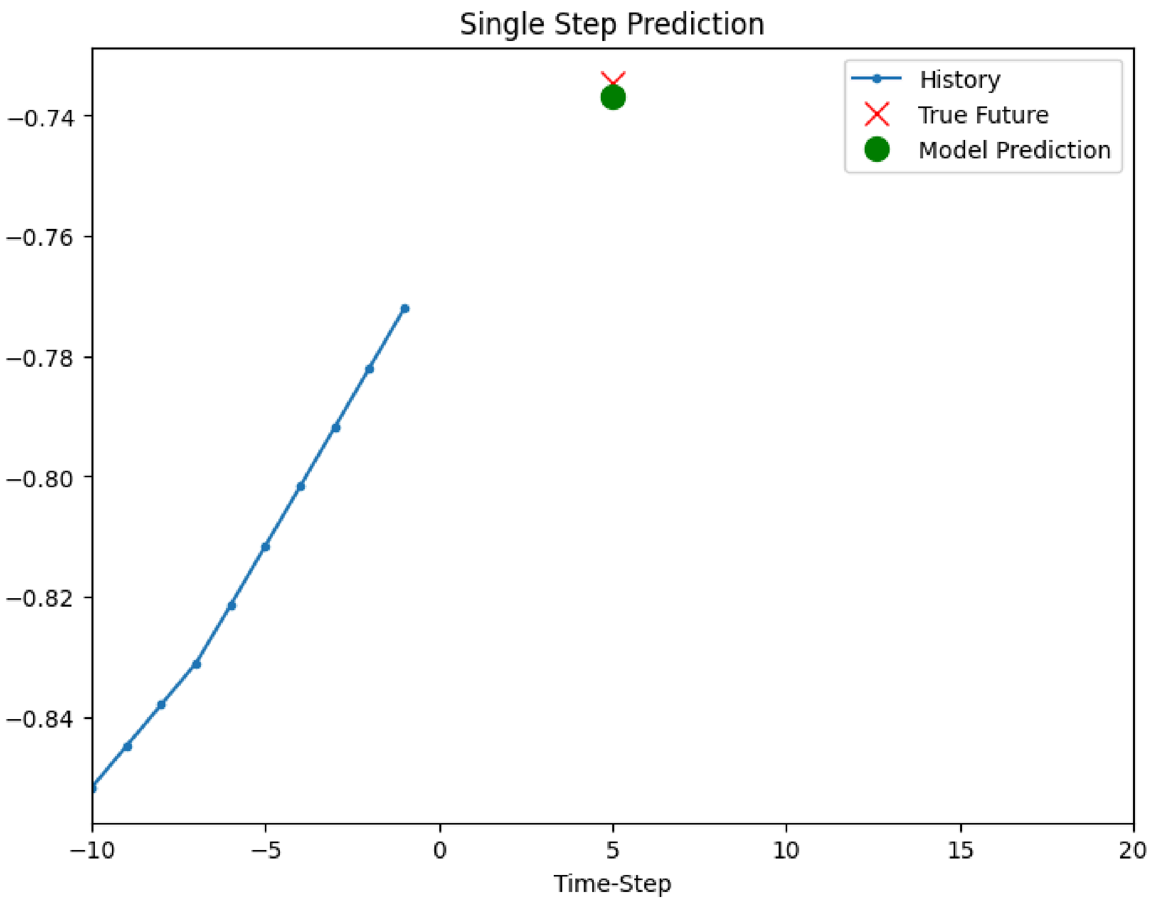
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```



8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step



8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step



Multivariate LSTM - Multiple Steps

Still, we use a series of observed values of the three variables "Infected", "Recovered", and "Deceased" (`past_history = 40, STEP =2`), but now to forcast the "Infected" values for a series day in the future (`future_target = 10`).

In [42]:
```
past_history = 40
future_target = 10
STEP =2
x_train_multi, y_train_multi = multivariate_data(dataset, dfInfected_data, 0, TR
                                                  past_history, future_target,
x_val_multi, y_val_multi = multivariate_data(dataset, dfInfected_data, TRAIN_SPL
                                              past_history, future_target, STE
```

In [43]:
```
print ('Single window of past history : {}'.format(x_train_multi[0].shape))
print ('\nTarget window to predict : {}'.format(y_train_multi[0].shape))
print ('\nNumber of traing data points: {}'.format(x_train_multi.shape[0]))
print ('\nNumber of test data points: {}'.format(x_val_multi.shape[0]))
```

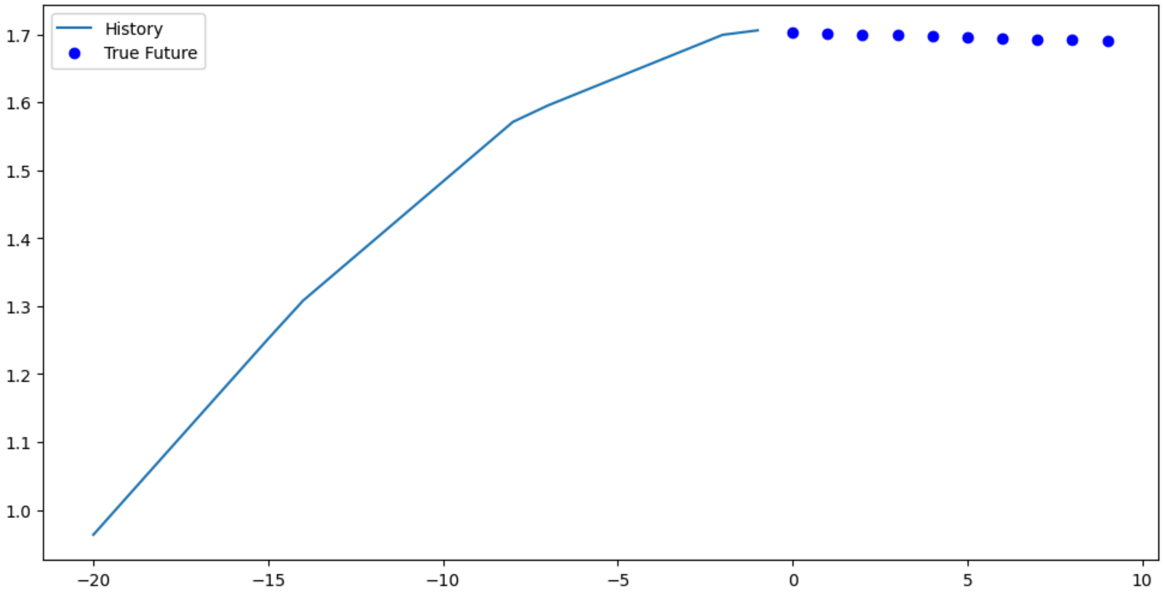
```
Single window of past history : (20, 3)

Target window to predict : (10,)

Number of traing data points: 60885

Number of test data points: 6765
```

As before, batching and resampling; the dataset is repeated indefinitely.

In [44]:
```
train_data_multi = tf.data.Dataset.from_tensor_slices((x_train_multi, y_train_mu
train_data_multi = train_data_multi.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZ

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 [45]:
```
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.plot(num_in, np.array(history[:, 0]), label='History')
    plt.plot(np.arange(num_out), np.array(true_future), 'bo', label='True Future
    if prediction.any():
        plt.plot(np.arange(num_out), np.array(prediction), 'ro', label='Predicte
    plt.legend(loc='upper left')
    plt.show()
```

In [46]:
```
for x, y in train_data_multi.take(1):
    multi_step_plot(x[0], y[0], np.array([0]))
```



Now we bild a model with two LSTM layers.

In [47]:
```
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(future_target))

multi_step_model.compile(optimizer=tf.keras.optimizers.RMSprop(clipvalue=1.0), l
multi_step_model.summary()
x_train_multi.shape[-2:]
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 20, 32)	4,608
lstm_3 (LSTM)	(None, 16)	3,136
dense_2 (Dense)	(None, 10)	170

Total params: 7,914 (30.91 KB)

Trainable params: 7,914 (30.91 KB)

Non-trainable params: 0 (0.00 B)

Out[47]: (20, 3)

In [48]:
```
for x, y in val_data_multi.take(1):
    print (multi_step_model.predict(x).shape)
```

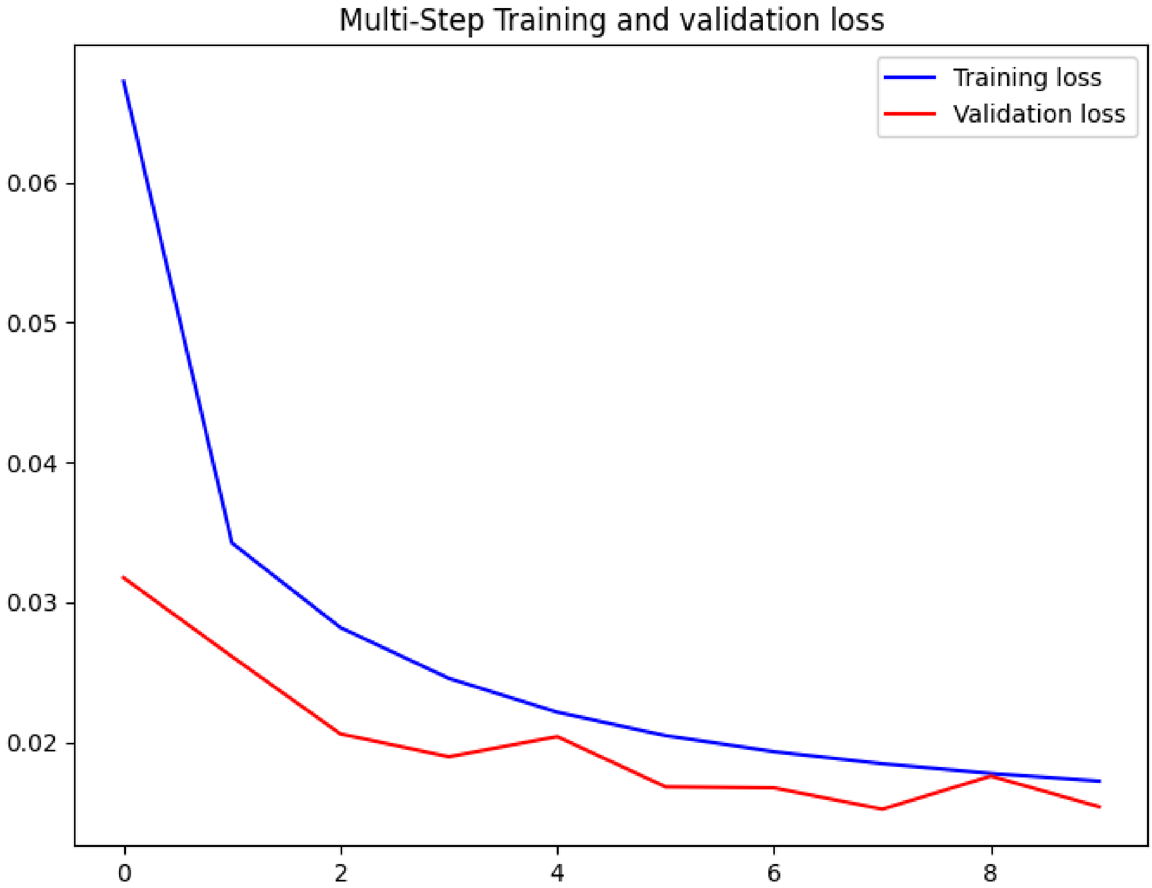
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
(256, 10)
```

The training time is longer for this more complex model.

In [49]:
```
multi_step_history = multi_step_model.fit(train_data_multi, epochs=EPOCHS,
                                          steps_per_epoch=EVALUATION_INTERVAL,
                                          validation_data=val_data_multi,
                                          validation_steps=50)
```

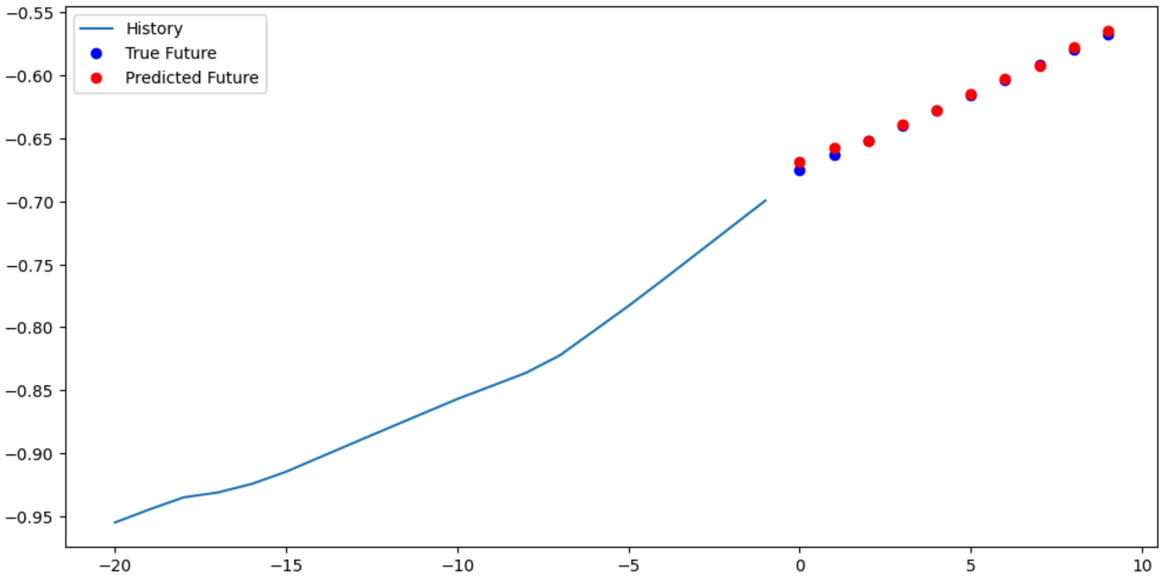
```
Epoch 1/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 41s 19ms/step - loss: 0.1224 - val_loss: 0.0317
Epoch 2/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 39s 20ms/step - loss: 0.0364 - val_loss: 0.0261
Epoch 3/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 39s 19ms/step - loss: 0.0293 - val_loss: 0.0206
Epoch 4/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - loss: 0.0253 - val_loss: 0.0190
Epoch 5/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 41s 21ms/step - loss: 0.0226 - val_loss: 0.0204
Epoch 6/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 43s 21ms/step - loss: 0.0209 - val_loss: 0.0168
Epoch 7/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 46s 23ms/step - loss: 0.0196 - val_loss: 0.0167
Epoch 8/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 48s 24ms/step - loss: 0.0186 - val_loss: 0.0152
Epoch 9/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 47s 23ms/step - loss: 0.0179 - val_loss: 0.0176
Epoch 10/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 45s 22ms/step - loss: 0.0174 - val_loss: 0.0154
```

In [50]:
```
plot_train_history(multi_step_history, 'Multi-Step Training and validation loss'
```

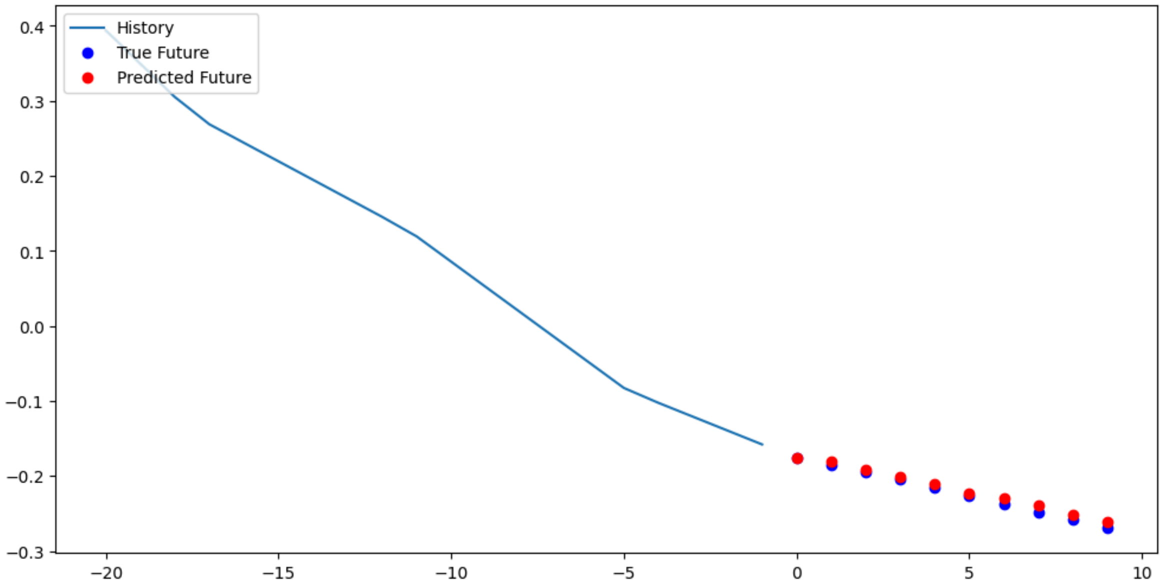


In [ ]:
```
for x, y in val_data_multi.take(3):
    multi_step_plot(x[0], y[0], multi_step_model.predict(x)[0])
```

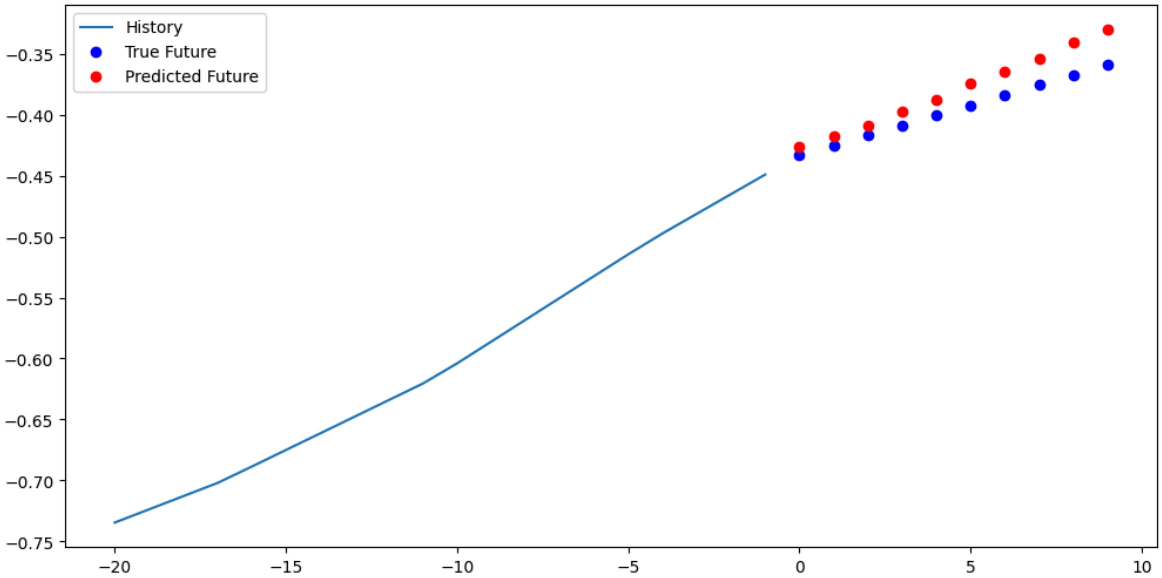
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```




```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```



8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step




```
The Kernel crashed while executing code in the current cell or a previous cell.

Please review the code in the cell(s) to identify a possible cause of the failur
e.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```