Time series forecasting

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

Steps

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

Imports and setup

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

Out[ ]:

	1	2	3	4	5	6	7	
0	100.287149	103.541223	95.879814	96.354848	96.980932	97.855310	98.940537	100.1874
1	0.993774	1.017558	1.070030	1.116168	1.142078	1.134735	1.182418	1.2723
2	0.000000	0.017741	0.036585	0.054735	0.074266	0.096065	0.117691	0.1391
3	0.000000	0.000178	0.000364	0.000562	0.000757	0.000947	0.001160	0.0013
4	103.489688	100.282780	96.634270	98.532514	99.089272	97.440900	98.416534	101.4049

5 rows × 501 columns

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

Out[ ]:

	1	2	3	4	5	6	7	
0	100.287149	103.541223	95.879814	96.354848	96.980932	97.855310	98.940537	100.1
4	103.489688	100.282780	96.634270	98.532514	99.089272	97.440900	98.416534	101.4
8	101.527421	97.711732	96.168179	95.677962	95.575326	96.109792	96.943831	98.0
12	101.061107	99.112815	106.651686	101.622904	97.726686	95.692173	97.438263	102.0
16	101.957189	101.898022	100.881113	99.892000	98.939878	98.048565	98.220024	99.2

5 rows × 501 columns

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

Out[ ]:

	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	1.35656
5	1.021677	1.045410	1.120324	1.175914	1.236878	1.306676	1.387931	1.477973	1.54987
9	1.020043	1.011238	1.031122	1.048642	1.049479	1.022891	1.035862	1.079177	1.11453
13	1.035248	1.014189	1.133178	1.135622	1.157984	1.213088	1.281406	1.359858	1.42323
17	1.012666	1.016949	1.053194	1.097599	1.143640	1.192369	1.238880	1.283688	1.32430

5 rows × 501 columns

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

Out[ ]:

	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	0.1
6	0.0	0.017909	0.035748	0.056118	0.076620	0.097338	0.119592	0.143024	0.171253	0.2
10	0.0	0.016990	0.034644	0.052866	0.071444	0.090609	0.108733	0.126058	0.142408	0.1
14	0.0	0.017002	0.036315	0.057484	0.078381	0.098831	0.119563	0.140509	0.166486	0.1
18	0.0	0.017589	0.037434	0.056572	0.076275	0.096907	0.116533	0.135387	0.157592	0.1

5 rows × 501 columns

In [ ]:

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

Out[ ]:

	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	0.0
7	0.0	0.000175	0.000351	0.000558	0.000763	0.000968	0.001196	0.001443	0.001719	0.0
11	0.0	0.000171	0.000352	0.000538	0.000729	0.000927	0.001126	0.001324	0.001488	0.0
15	0.0	0.000181	0.000364	0.000563	0.000774	0.001003	0.001192	0.001351	0.001575	0.0
19	0.0	0.000180	0.000358	0.000550	0.000740	0.000931	0.001138	0.001359	0.001574	0.0

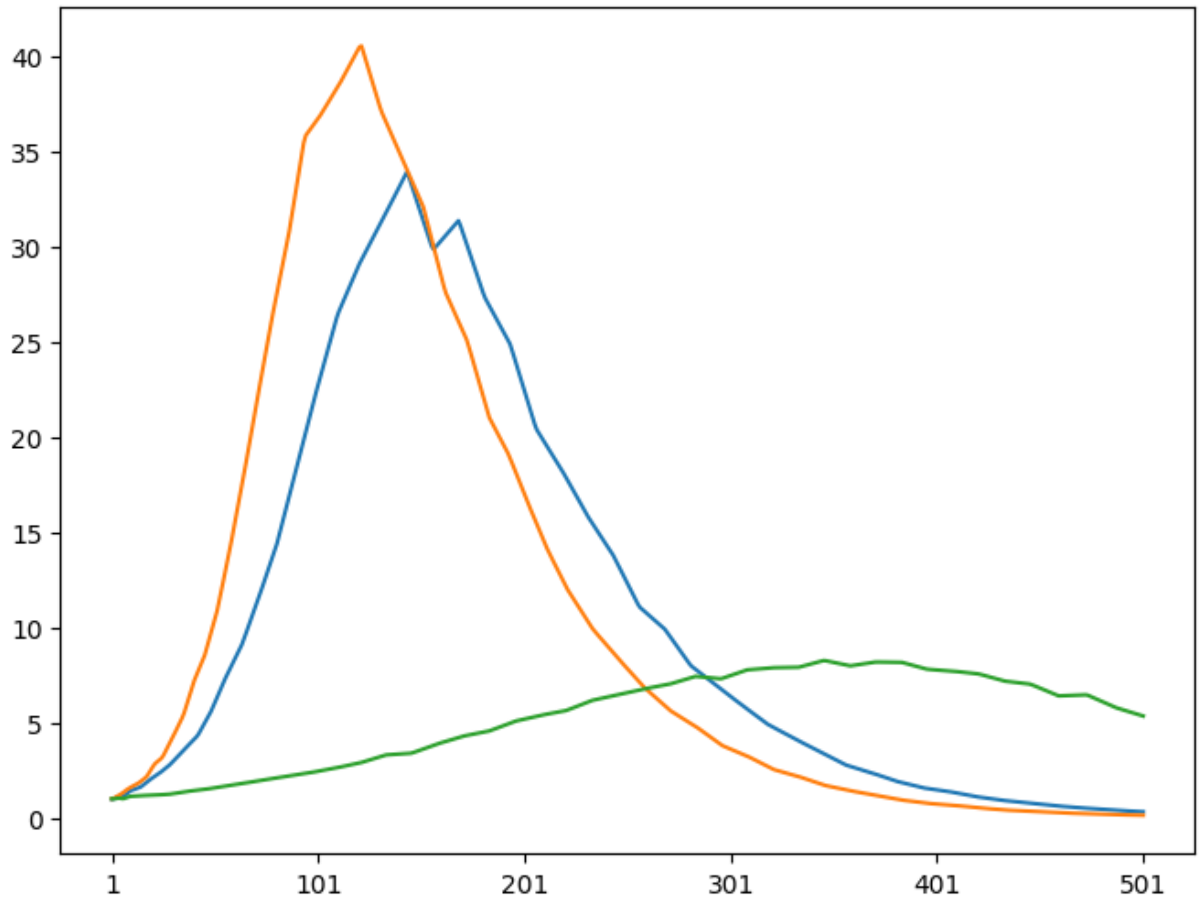
5 rows × 501 columns

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

In [ ]:

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

Out[ ]: <AxesSubplot:>



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

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

Out[ ]:
```
135
```

We standardize the data.

In [ ]:
```
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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
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
```

Create test data where the infection reached the peak

In [ ]:
```
peak_idx = np.argmax(uni_data, axis=1)
peak_uni = []
for i in range(len(uni_data)):
    idx = peak_idx[i]
    start = max(idx - univariate_past_history, 0)
    end = min(idx + univariate_past_history, uni_data.shape[1])
    temp_peak_uni = []
    for j in range(start, end+1):
        temp_peak_uni.append(uni_data[i,j])
    peak_uni.append(np.array(temp_peak_uni))

peak_uni = np.array(peak_uni)
```

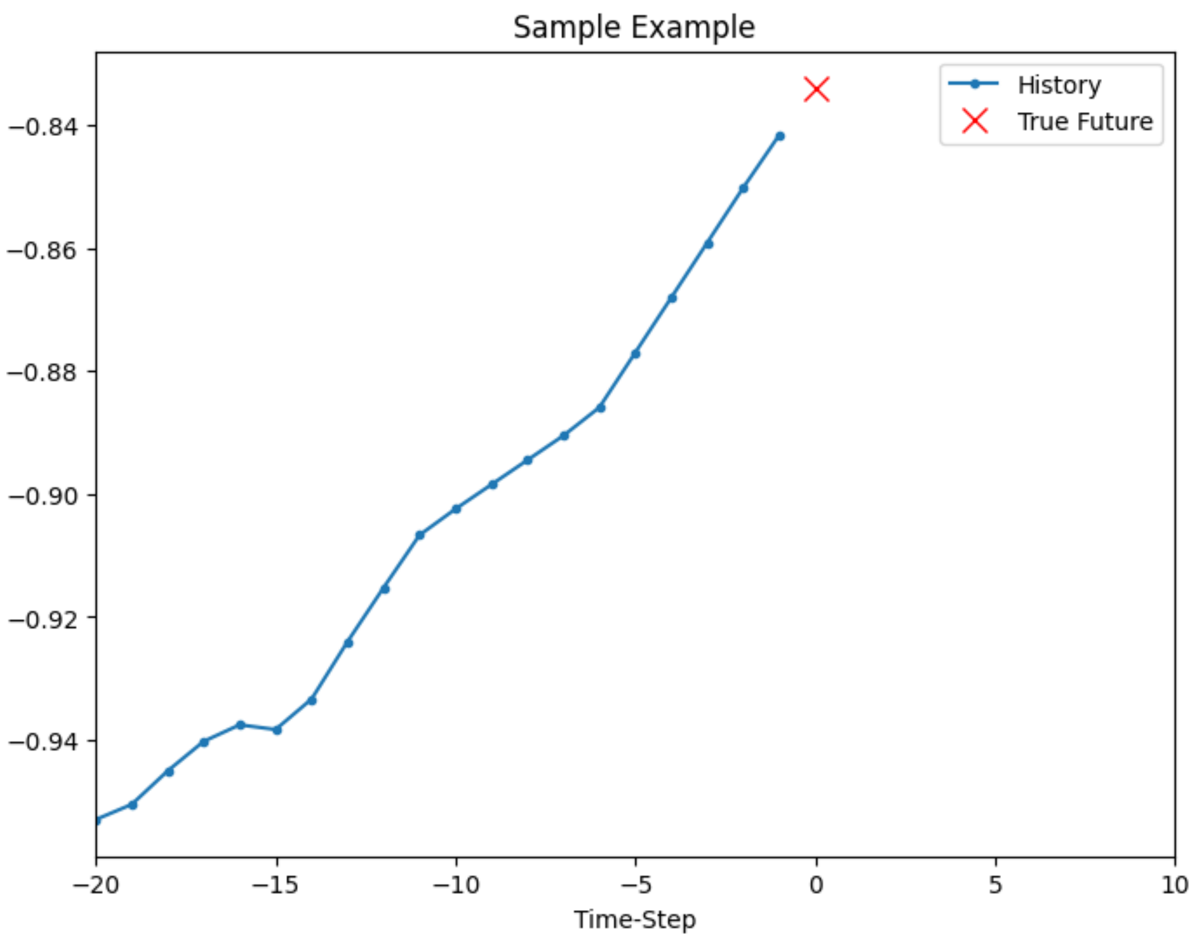
In [ ]:
```
x_peak_uni, y_peak_uni = univariate_data(peak_uni, 0, len(uni_data),
                                         univariate_past_history,
                                         univariate_future_target)
```

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

In [ ]:
```
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[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
```

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

Out[ ]:
```
<module 'matplotlib.pyplot' from '/media/home/ngundermann/workspace/DML_notebooks/v
env/lib/python3.10/site-packages/matplotlib/pyplot.py'>
```



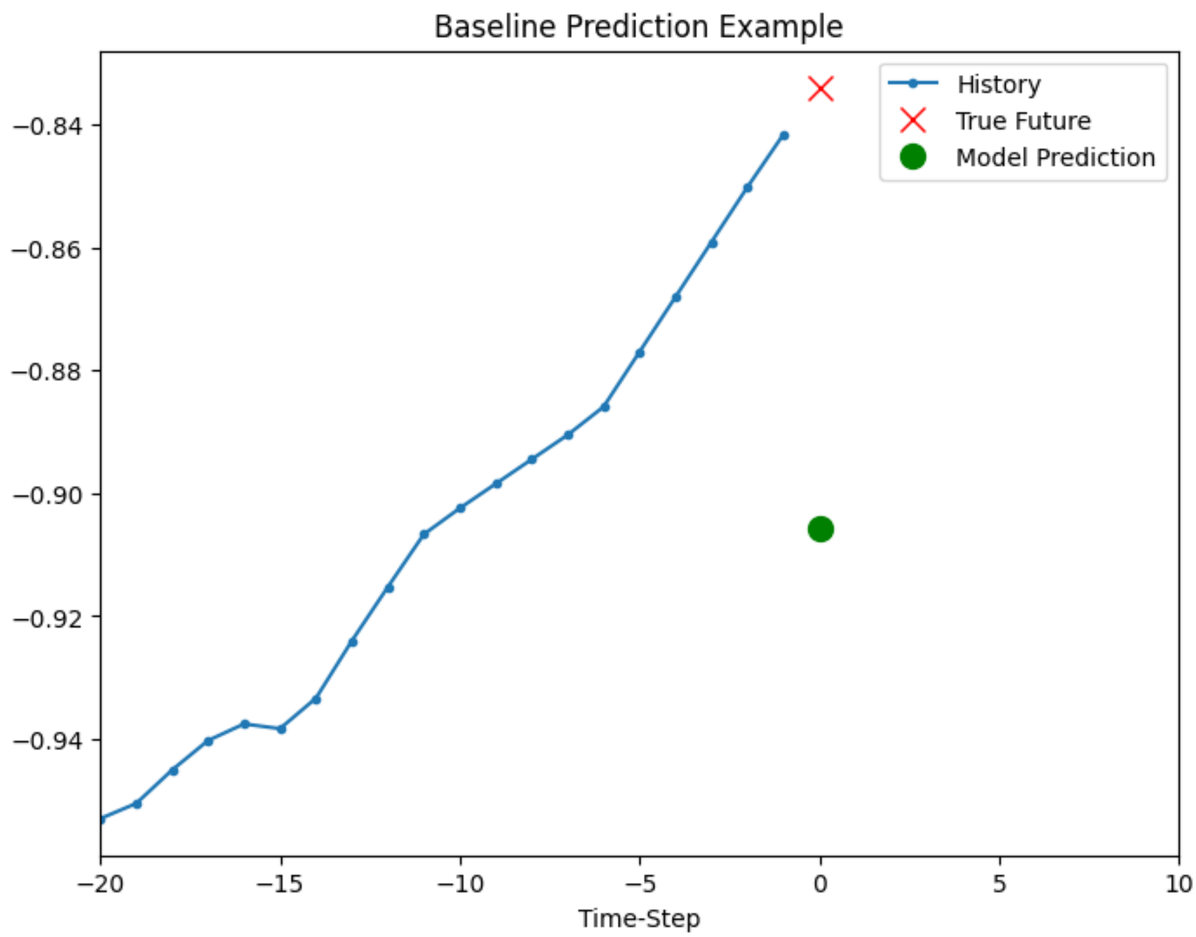
Baseline forecasting

Predicts the mean of the `history` .

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

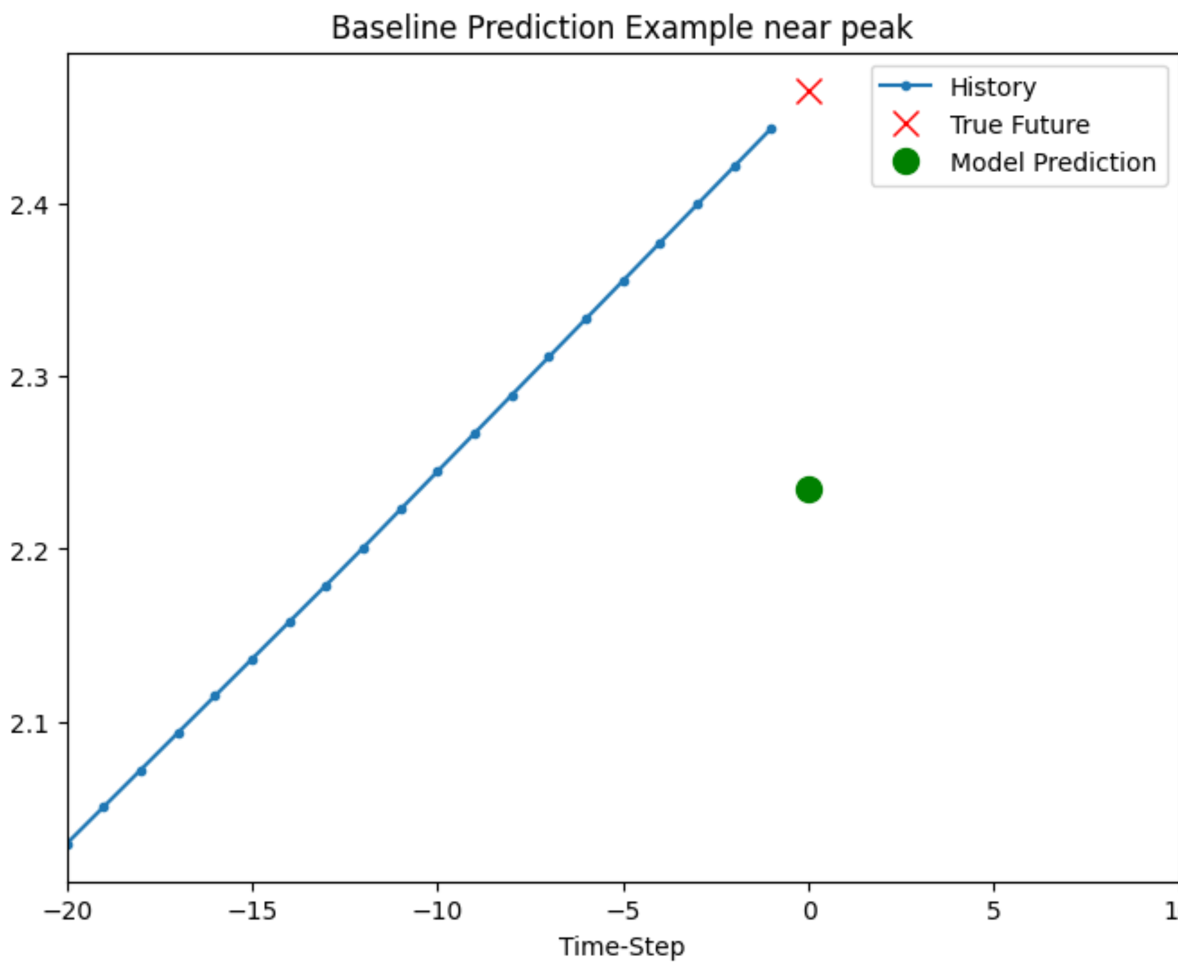
In [ ]:
```
show_plot([x_train_uni[0], y_train_uni[0], baseline(x_train_uni[0])], 0, 'Baseline
```

Out[ ]:
```
&lt;module 'matplotlib.pyplot' from '/media/home/ngundermann/workspace/DML_notebooks/v
env/lib/python3.10/site-packages/matplotlib/pyplot.py'&gt;
```



In [ ]:
```
show_plot([x_peak_uni[0], y_peak_uni[0], baseline(x_peak_uni[0])], 0, 'Baseline Pre
```

Out[ ]:
```
&lt;module 'matplotlib.pyplot' from '/media/home/ngundermann/workspace/DML_notebooks/v
env/lib/python3.10/site-packages/matplotlib/pyplot.py'&gt;
```



Univariate *GRU* based forecasting

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

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

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

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

In [ ]:
```
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_SIZE).

val_univariate = tf.data.Dataset.from_tensor_slices((x_val_uni, y_val_uni))
val_univariate = val_univariate.batch(BATCH_SIZE).repeat()

peak_val_univariate = tf.data.Dataset.from_tensor_slices((x_peak_uni, y_peak_uni))
peak_val_univariate = peak_val_univariate.batch(1).repeat()

train_univariate
```

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

We define the first *GRU* model with 8 units.

In [ ]:
```
simple_lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.GRU(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:]
```

```
/media/home/ngundermann/workspace/DML_notebooks/venv/lib/python3.10/site-packages/ke
ras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
gru_12 (GRU)	(None, 8)	264
dense_9 (Dense)	(None, 1)	9

Total params: 273 (1.07 KB)

Trainable params: 273 (1.07 KB)

Non-trainable params: 0 (0.00 B)

Out[ ]:
```
(20, 1)
```

The GRU layer comes with less parameters than the LSTM layer (320) we have seen in the original notebook. This is becaust the GRU architecture is much simpler than LSTMs'.

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

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
(256, 1)
(256,)
2024-05-30 09:00:21.924564: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

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

In [ ]:
```python
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)
```

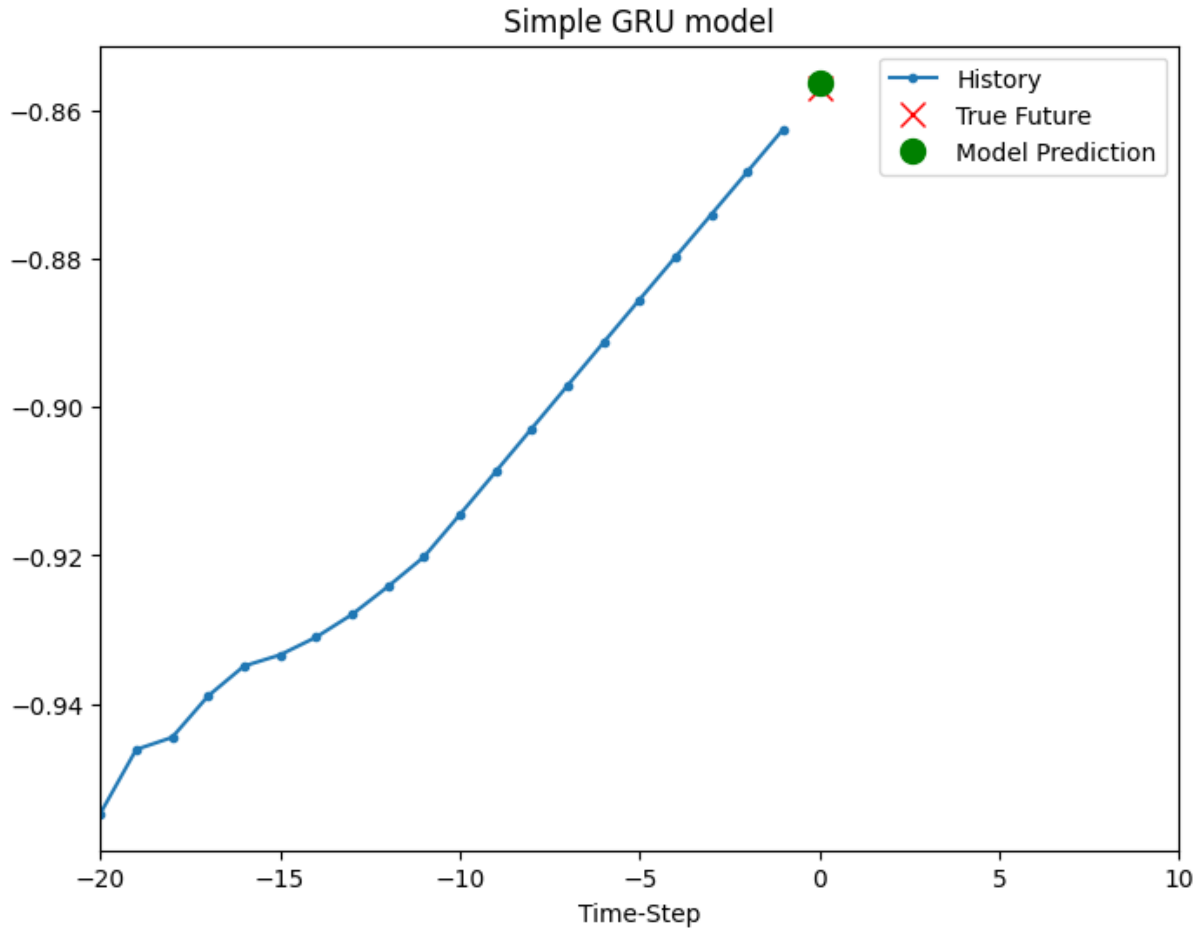
```
Epoch 1/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - loss: 0.0792 - val_loss: 0.0031
Epoch 2/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0029 - val_loss: 0.0026
Epoch 3/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0024 - val_loss: 0.0016
Epoch 4/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0022 - val_loss: 0.0014
Epoch 5/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0020 - val_loss: 0.0015
Epoch 6/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0019 - val_loss: 0.0013
Epoch 7/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0017 - val_loss: 0.0012
Epoch 8/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0017 - val_loss: 0.0018
Epoch 9/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - loss: 0.0017 - val_loss: 8.7684e-04
Epoch 10/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - loss: 0.0017 - val_loss: 0.0012
```

Out[ ]: `<keras.src.callbacks.history.History at 0x7f04384e0e50>`

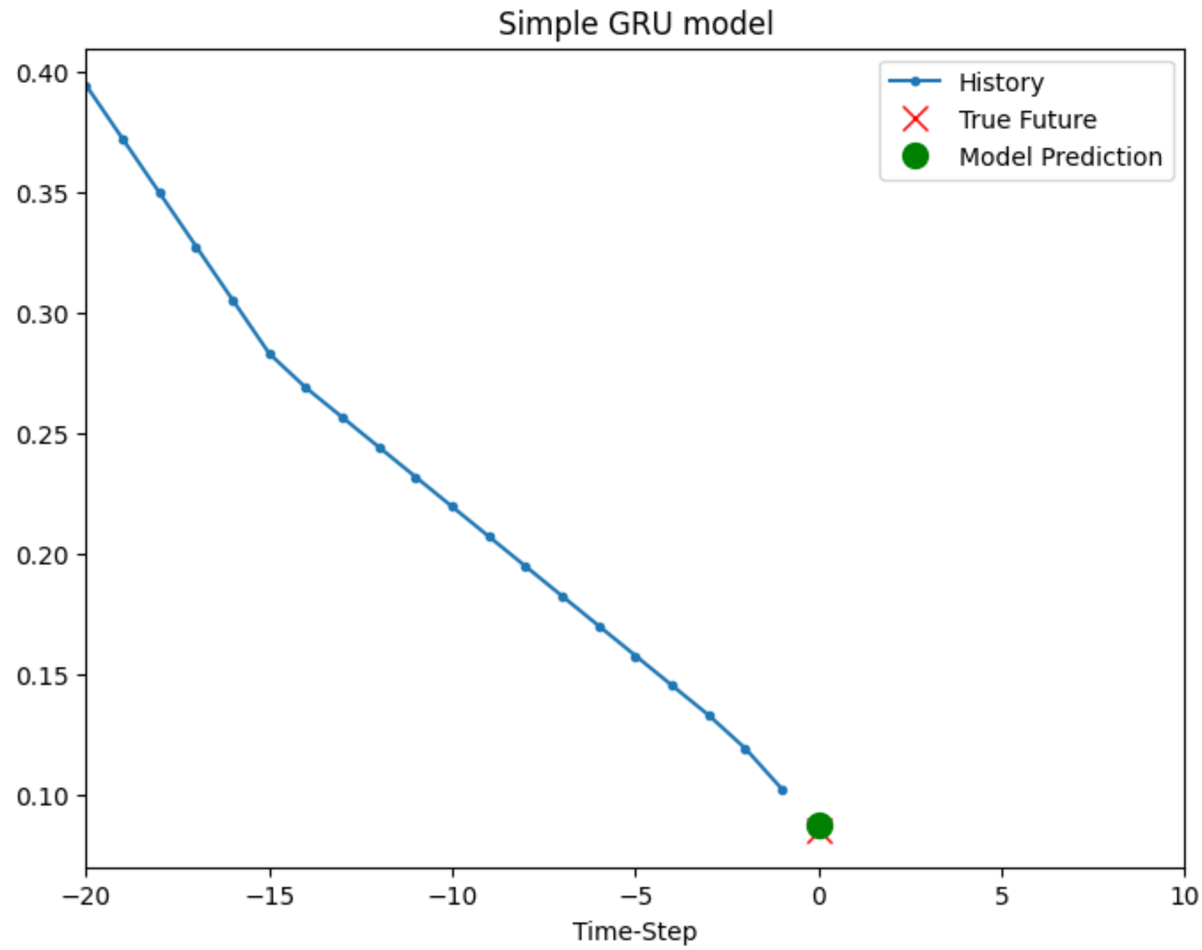
The training loss went down very fast and convereged after a few epochs. However, the validation loss stated very low at the beginning, but did decrease quite fast. The training of one epoch finished faster, which could be caused by the fewer parameters of the model but also by the different machine on that the training was ran.

In [ ]:
```python
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()
```

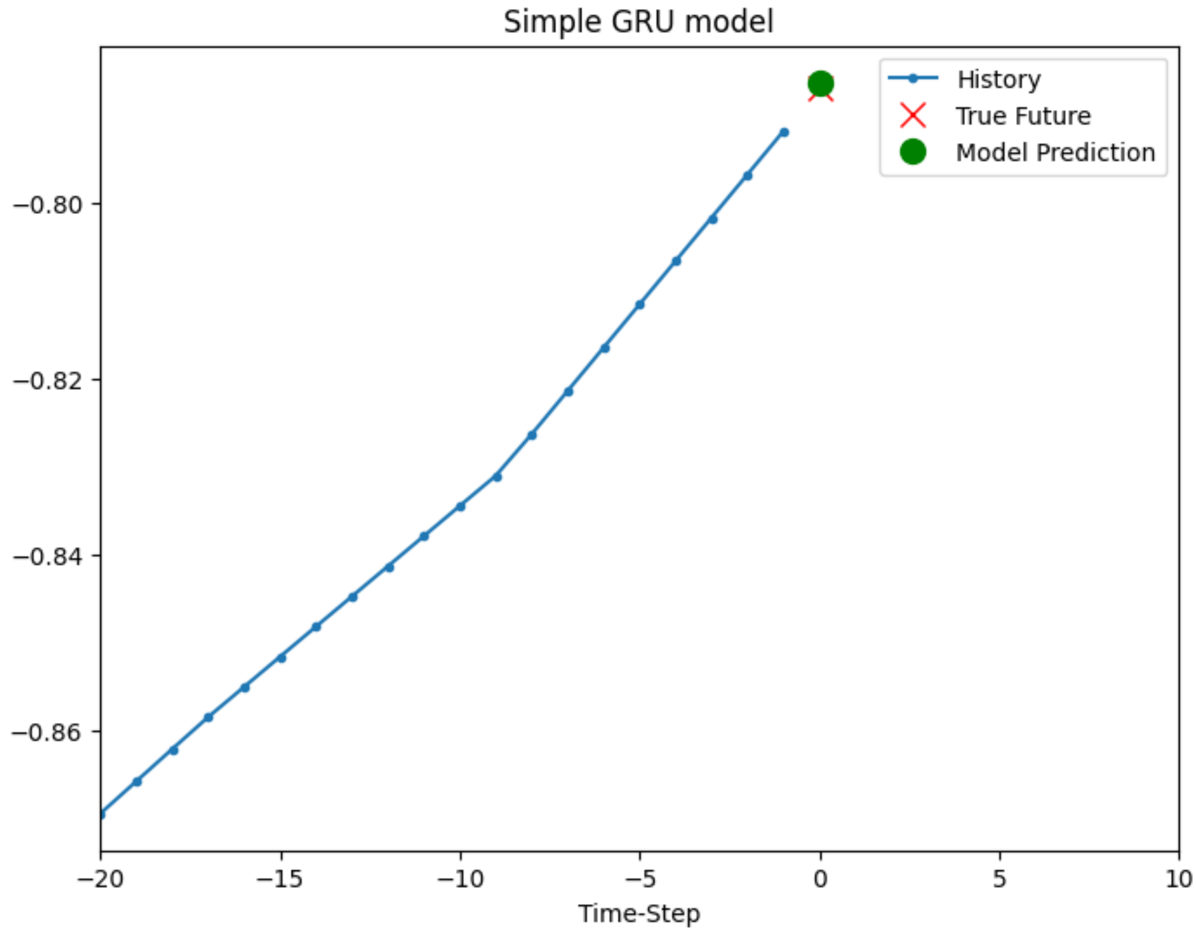
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



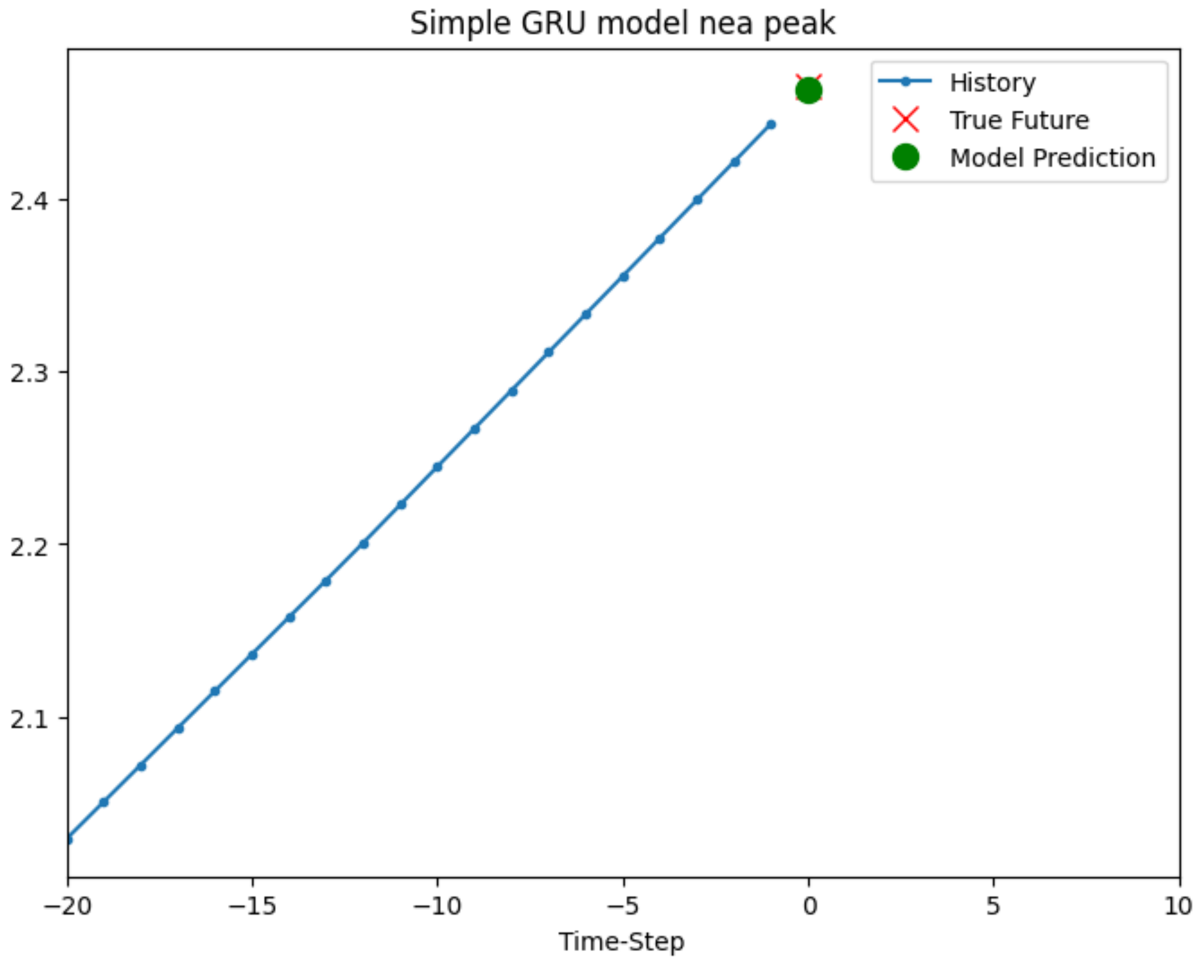

```
2024-05-30 09:01:29.437318: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

The predictions look quite good, since the validataion loss was quite low the whole time. The prediction are not exactly at the true value, but very close to it.

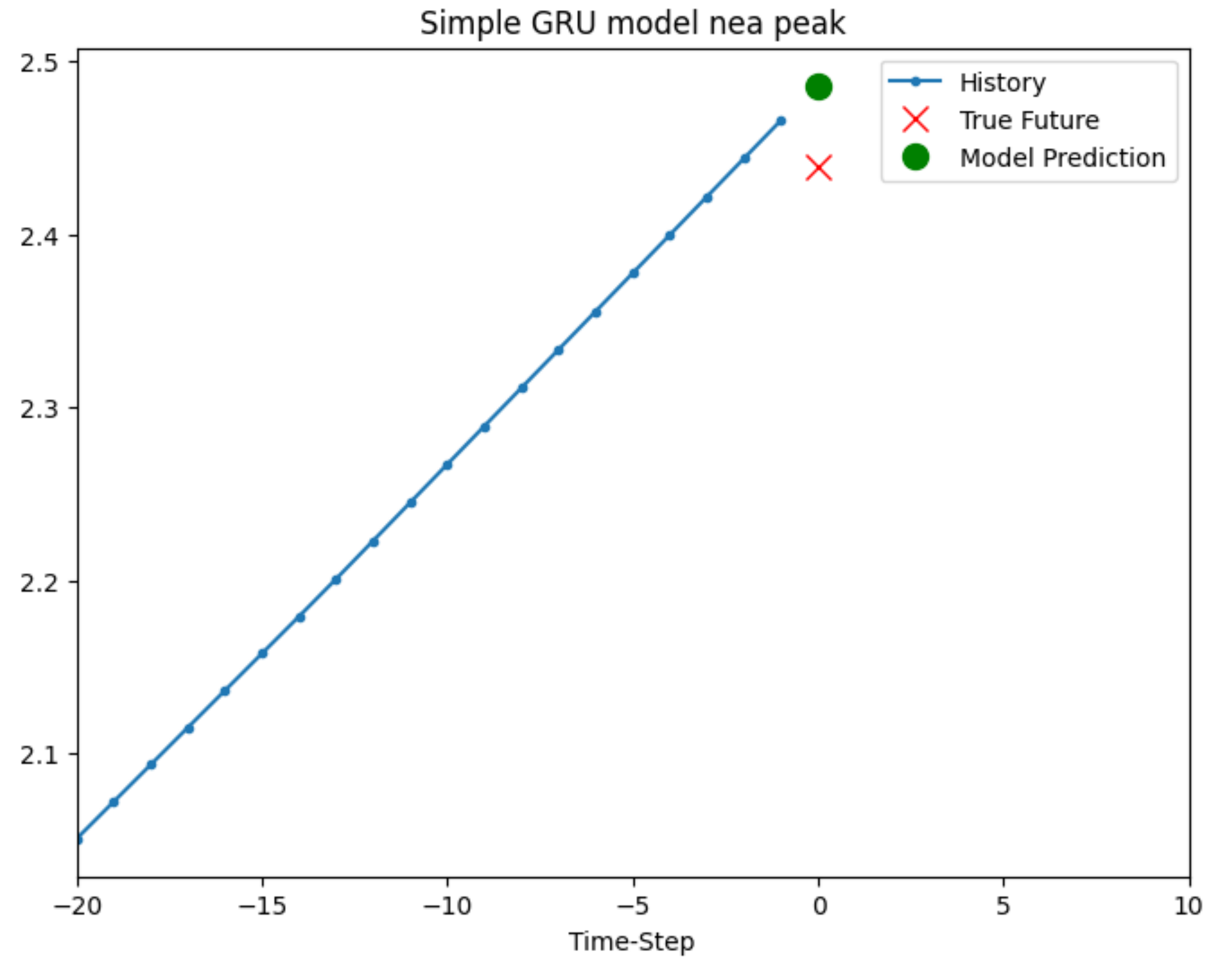
Diagrams with the peek of infection

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

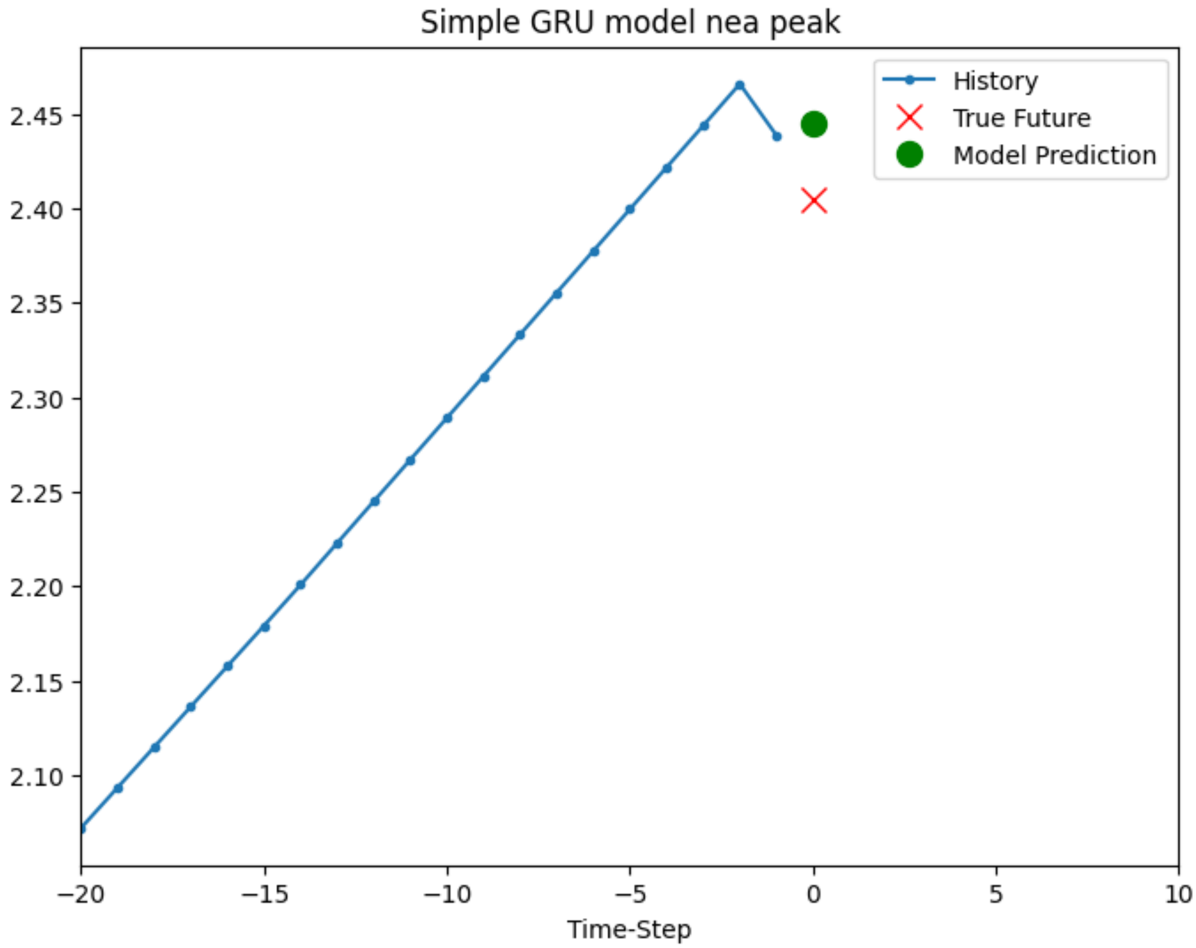
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```




```
2024-05-30 09:01:29.878153: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

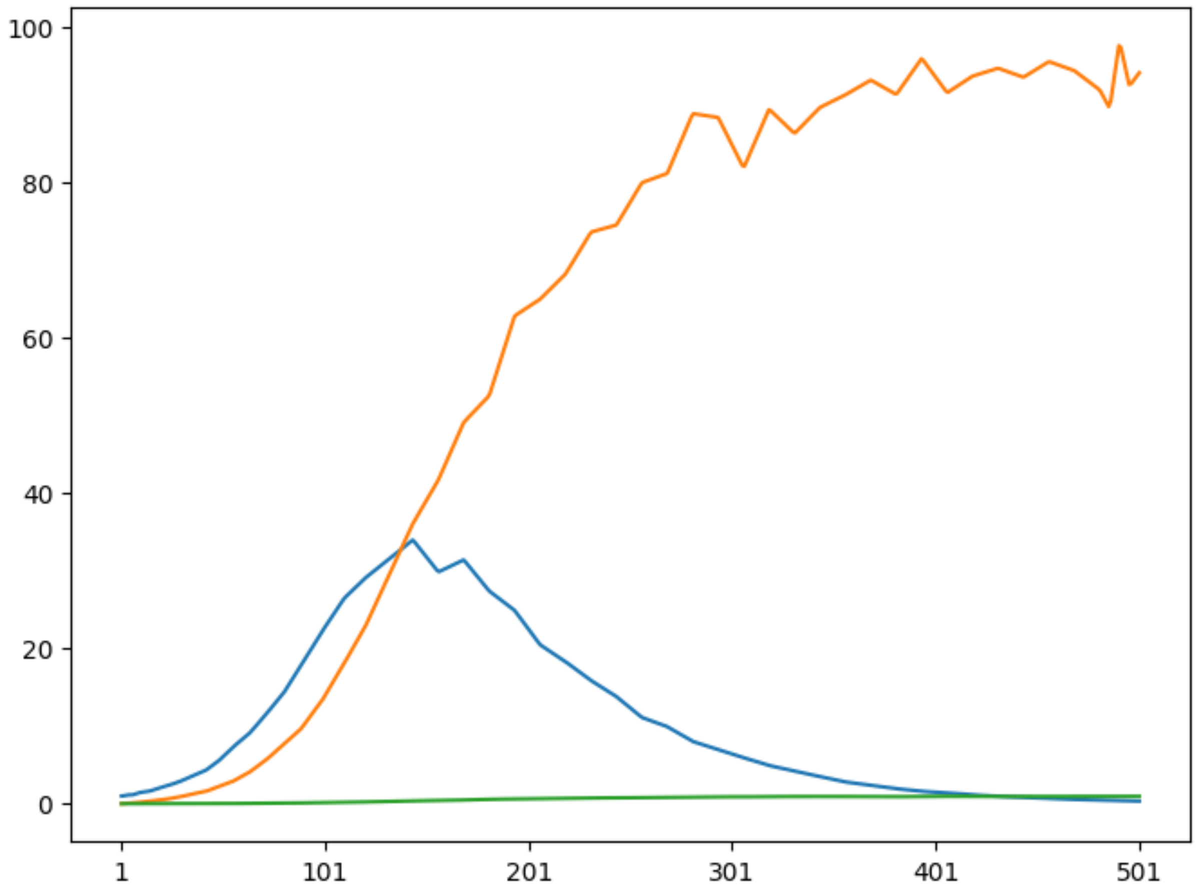
Even the prediction near infection peak were not that good. It seems that the model has problems with changing slopes of the ground truths trajectory, i.e. when the infection starts to decline.

Multivariate *GRU* 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 [ ]:
```
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 [ ]:
```
#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_std
#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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
past_history = 20
future_target = 5
STEP = 2

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

In [ ]:
```
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, creating a dataset around the infection peak.

In [ ]:
```
def multivariate_data_near_peak_infection(dataset, past_history, future_target):
    peak_idx = np.argmax(dataset[0], axis=1)
    peak_infected = []
    peak_infected_recovered = []
    peak_infected_dead = []
    for i in range(len(dataset[1])):
        idx = peak_idx[i]
        start = max(idx - 1 - past_history - future_target // 2 , 0)
        end = min(idx + future_target // 2 + future_target % 2, dataset.shape[2])
        temp_peak_infected = []
        temp_peak_infected_recovered = []
        temp_peak_infected_dead = []
        for j in range(start, end):
            temp_peak_infected.append(dataset[0,i,j])
            temp_peak_infected_recovered.append(dataset[1,i,j])
            temp_peak_infected_dead.append(dataset[2,i,j])
        peak_infected.append(np.array(temp_peak_infected))
        peak_infected_recovered.append(np.array(temp_peak_infected_recovered))
        peak_infected_dead.append(np.array(temp_peak_infected_dead))

    return np.array([peak_infected, peak_infected_recovered, peak_infected_dead])
```

In [ ]:
```
peak_data = multivariate_data_near_peak_infection(dataset, past_history, future_tar
peak_data.shape
```

Out[ ]: (3, 150, 26)

In [ ]:
```
x_peak_single, y_peak_single = multivariate_data(peak_data, peak_data[0], 0, peak_d
                                               past_history, future_target, STEP,
                                               single_step=True)
```

In [ ]:
```
x_peak_single.shape
```

Out[ ]: (150, 10, 3)

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

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

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()

peak_data_single = tf.data.Dataset.from_tensor_slices((x_peak_single, y_peak_single
peak_data_single = peak_data_single.batch(1).repeat()
```

In [ ]:
```
single_step_model = tf.keras.models.Sequential()
single_step_model.add(tf.keras.layers.GRU(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_model.summary()
x_train_single.shape[-2:]
```

```
/media/home/ngundermann/workspace/DML_notebooks/venv/lib/python3.10/site-packages/ke
ras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
gru_13 (GRU)	(None, 32)	3,552
dense_10 (Dense)	(None, 1)	33

Total params: 3,585 (14.00 KB)

Trainable params: 3,585 (14.00 KB)

Non-trainable params: 0 (0.00 B)

Out[ ]: (10, 3)

Again we got a network with fewer parameters, since the GRU architecture comes with less parameters than the LSTM architecture.

In [ ]:
```python
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 1ms/step
(256, 1)

 Number of traing data points
64260

 Number of test data points
7140
2024-05-30 09:01:30.823578: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

In [ ]:
```python
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)
```

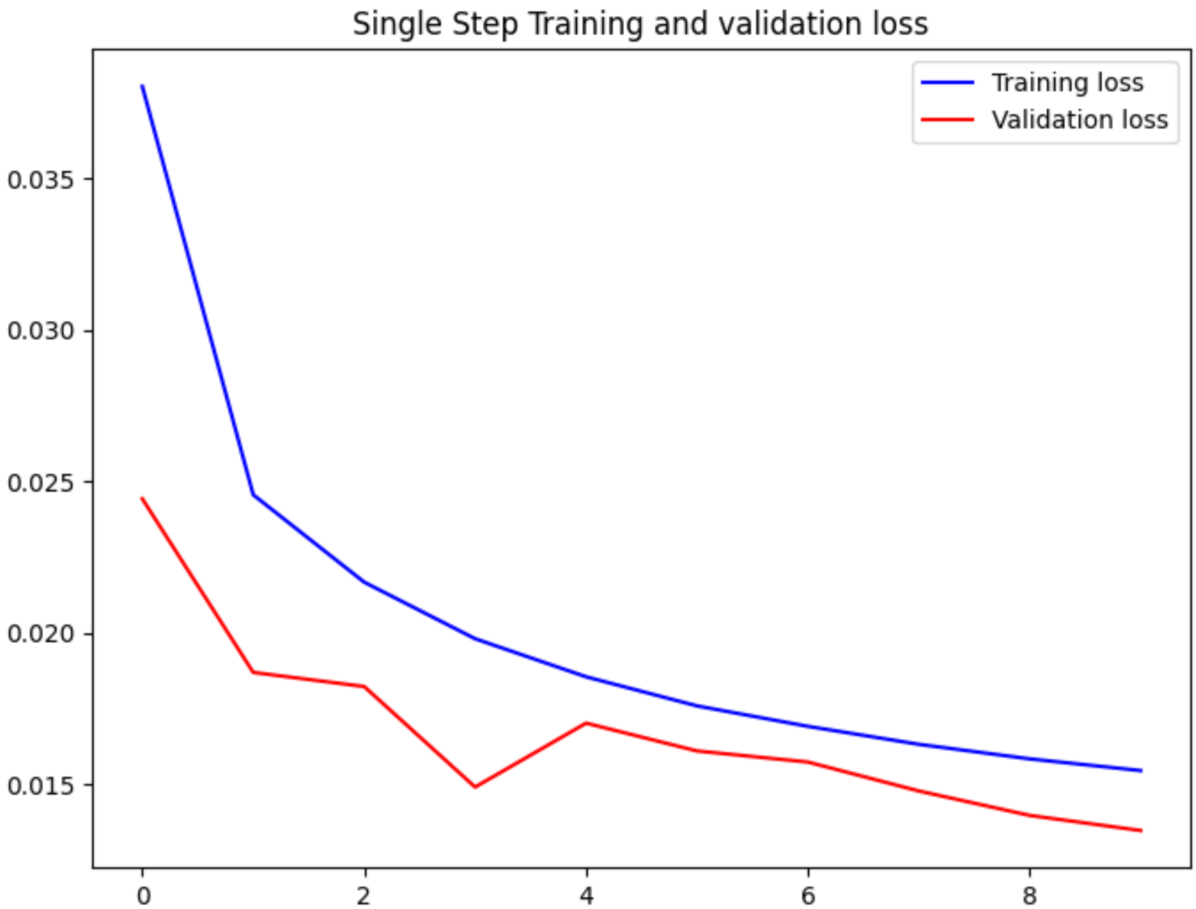
```
Epoch 1/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - loss: 0.0670 - val_loss: 0.0244
Epoch 2/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0256 - val_loss: 0.0187
Epoch 3/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0222 - val_loss: 0.0182
Epoch 4/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0202 - val_loss: 0.0149
Epoch 5/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0188 - val_loss: 0.0170
Epoch 6/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0177 - val_loss: 0.0161
Epoch 7/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0170 - val_loss: 0.0157
Epoch 8/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0164 - val_loss: 0.0148
Epoch 9/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0159 - val_loss: 0.0140
Epoch 10/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 0.0155 - val_loss: 0.0135
```

In [ ]:

```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 [ ]:

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

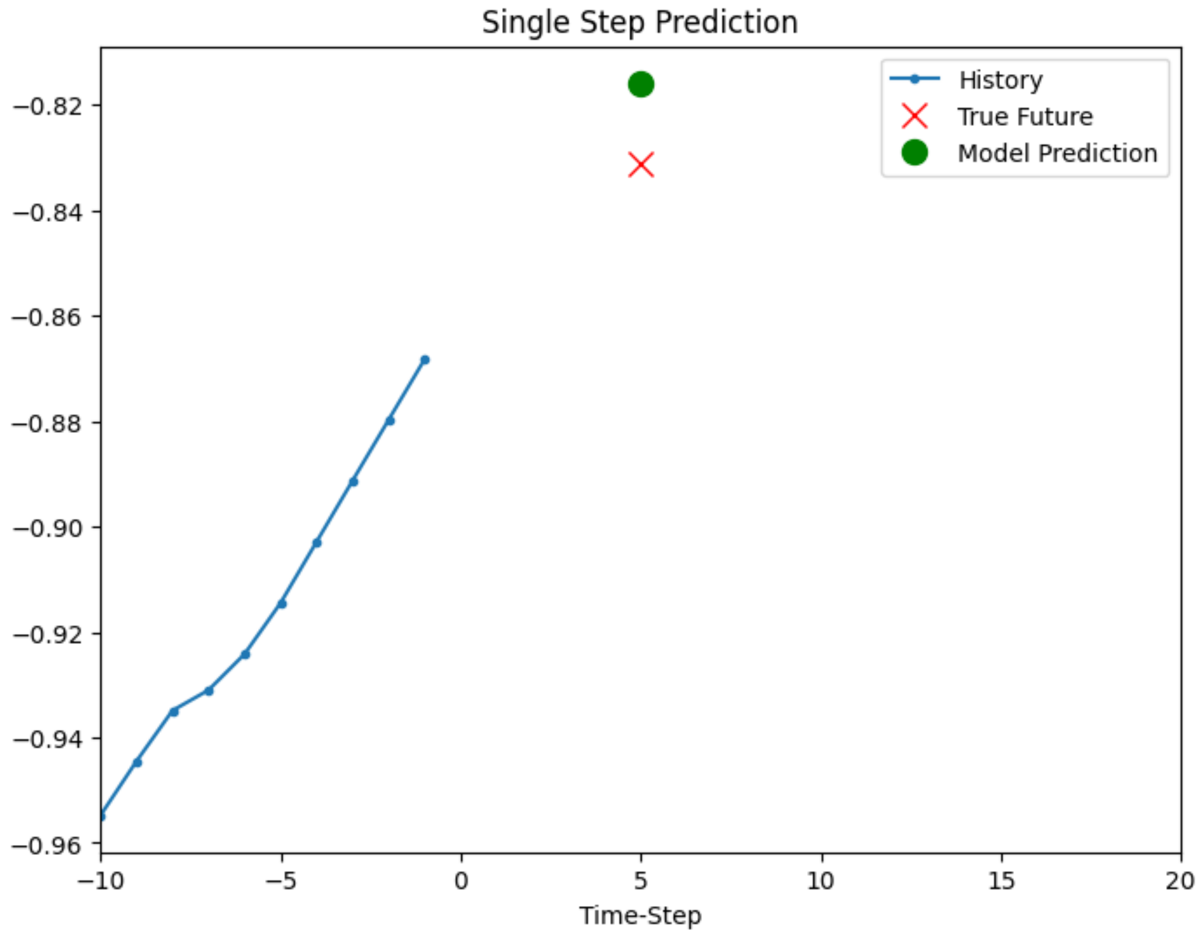


Here, we see once again a big decrease in training loss at the very beginning. However, here, the validation loss tend to shrink next to the training loss, which could indicate that the resulting network will not suffer from overfitting and is, thus, more general compared to the previous one. Moreover, I think the training needs some more epochs in order to let the loss converge at a certain value. This was probably not reached after 10 epochs.

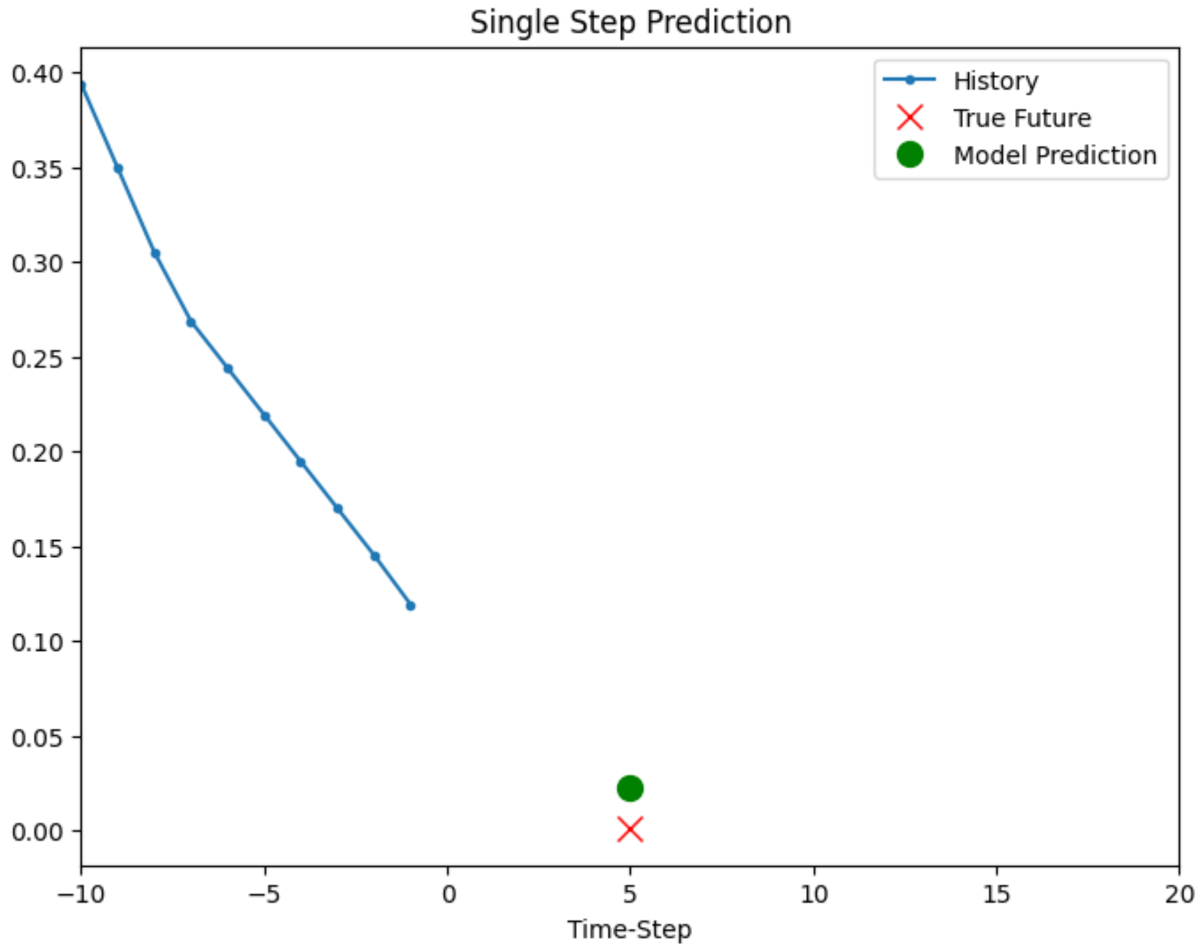
According to time, once again it was a little bit fast than the training of the model from the original notebook, which could be due to the architecture coming with fewer parameters but also to the different machine on that the training was ran.

In [ ]:
```
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()
```

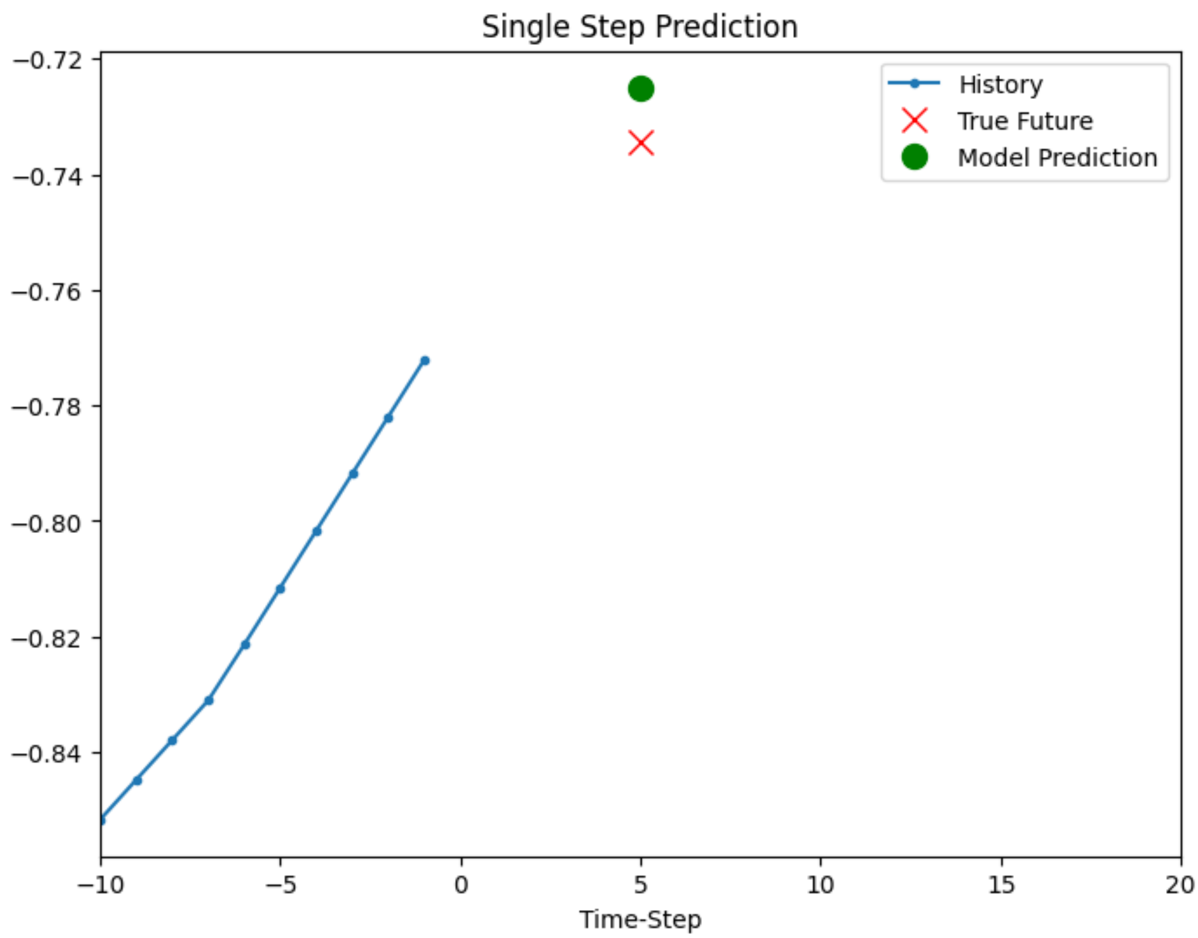
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



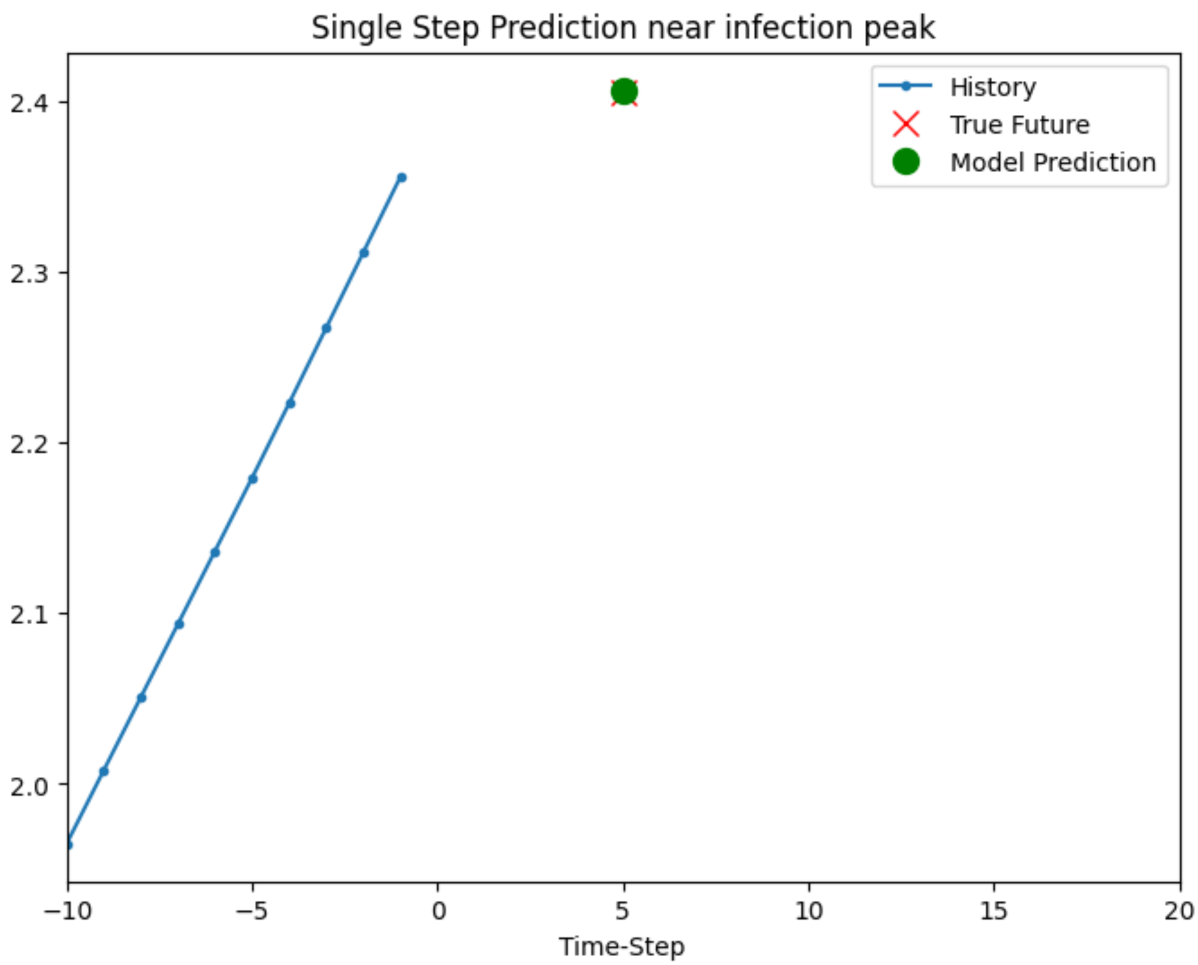

```
2024-05-30 09:02:40.198774: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

Looking at the predictions they a a bit worse compared to the previous model. Here, we have to keep in mind, that both models performed different predicitons. Here, we wanted to predict the 5 day future, while with the first model we wanted to predict the 1 day future. So maybe the error, we have seen in the previous model is likely to add up when the number of days we want to forcast into the future increases. Anyway, that the prediction is worse than in the preivous model, could have been noticed when comparing the validation loss of the models.

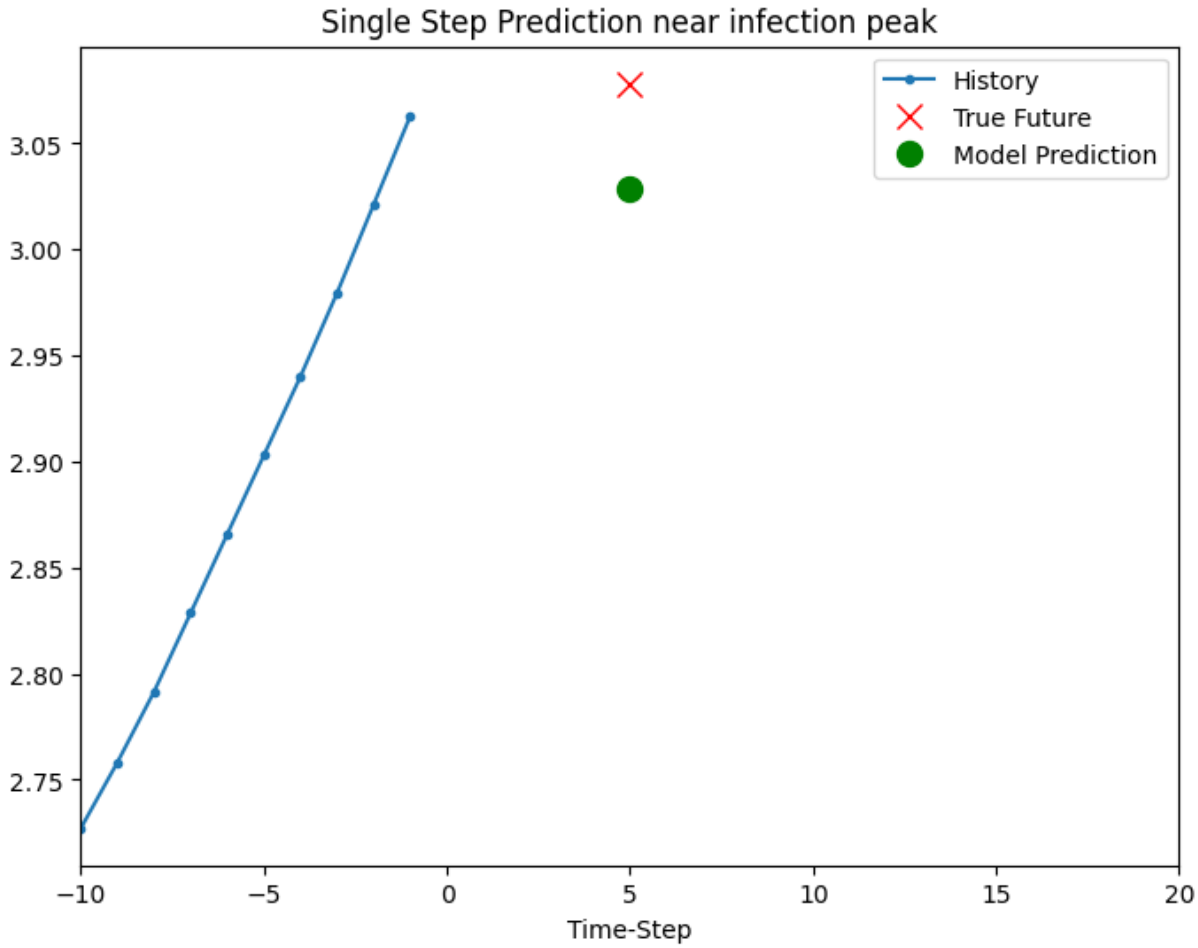
Now, let's plot some predictions near the infection peak.

In [ ]:
```
for x, y in peak_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 near infection peak')
    plot.show()
```

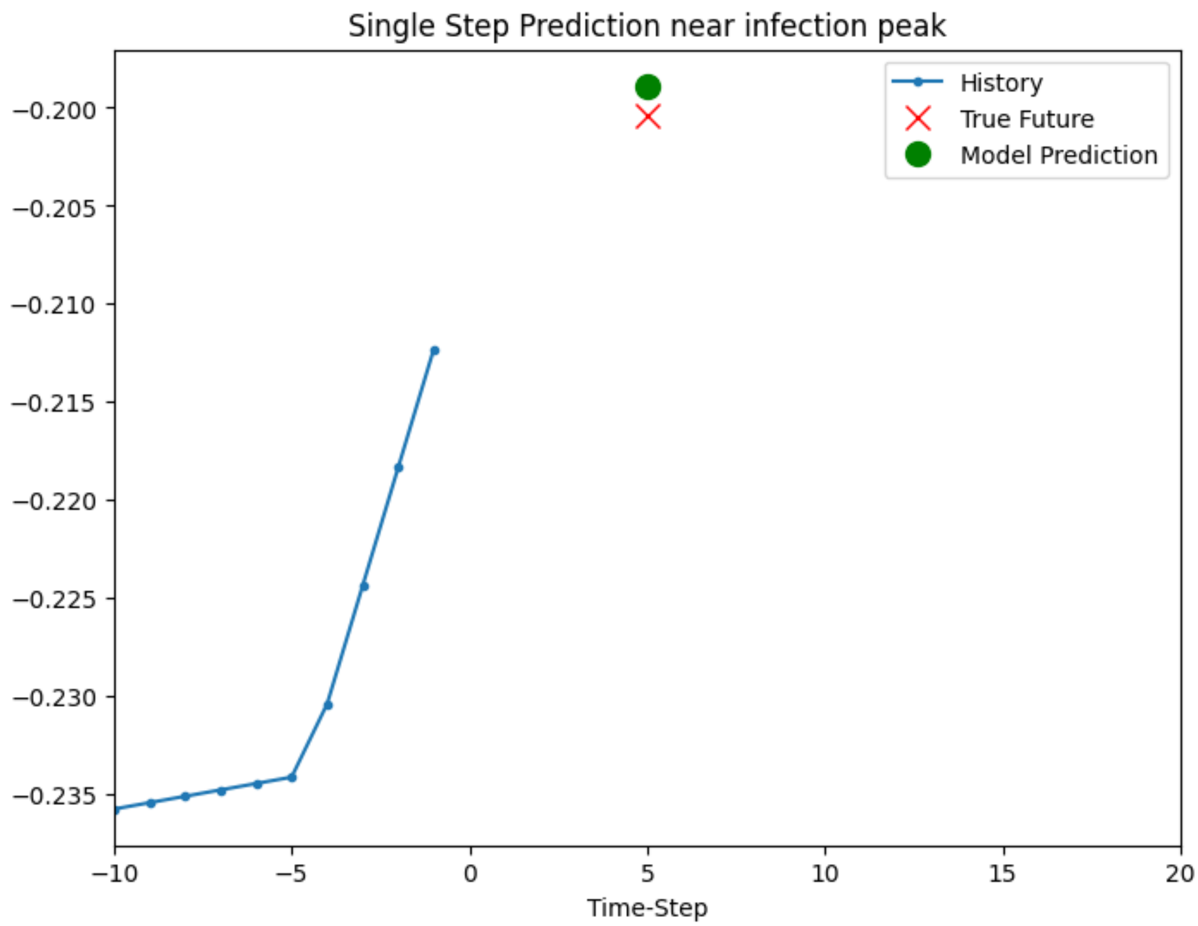
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```




```
2024-05-30 09:02:40.919216: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

Here, we should take into account that the prediction made is for 3 days after the peak infection. So it is likely that the ground truth is declining in the time stamp to be predicted. Therefore, this model seems to encounter the same problem with making predictions after the direction of the trend has changed, since the second predicitons is a way off compared to the ture value. Maybe example 1 and 3 seems to be better, because there is some kind of platoe in infection.

Multivariate *GRU* - 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 [ ]:
```
past_history = 40
future_target = 10
STEP =2
x_train_multi, y_train_multi = multivariate_data(dataset, dfInfected_data, 0, TRAIN
                                                 past_history, future_target, ST
x_val_multi, y_val_multi = multivariate_data(dataset, dfInfected_data, TRAIN_SPLIT,
                                             past_history, future_target, STEP)
```

In [ ]:
```
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, retrieve data points near infection peak.

In [ ]:
```
peak_data = multivariate_data_near_peak_infection(dataset, past_history, future_tar
peak_data.shape
```

Out[ ]: (3, 150, 51)

In [ ]:
```
x_peak_multi, y_peak_multi = multivariate_data(peak_data, peak_data[0], 0, peak_dat
                                               past_history, future_target, STEP)
```

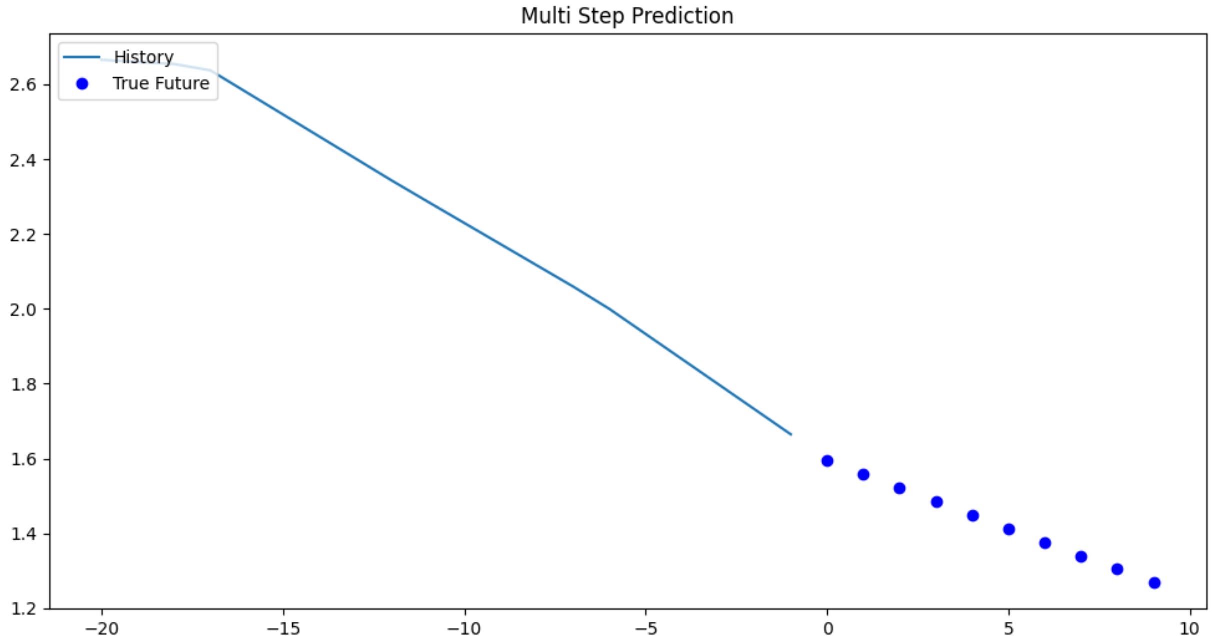
As before, batching and resampling; the dataset is repeated indefinitely.

In [ ]:
```
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).

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()

peak_data_multi = tf.data.Dataset.from_tensor_slices((x_peak_multi, y_peak_multi))
peak_data_multi = peak_data_multi.batch(1).repeat()
```

In [ ]:
```
def multi_step_plot(history, true_future, prediction, title='Multi Step 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='Predicted F
    plt.legend(loc='upper left')
    plt.title(title)
    plt.show()
```

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

```
2024-05-30 09:02:41.757811: W tensorflow/core/kernels/data/cache_dataset_ops.cc:858]
The calling iterator did not fully read the dataset being cached. In order to avoid
unexpected truncation of the dataset, the partially cached contents of the dataset
will be discarded. This can happen if you have an input pipeline similar to `datase
t.cache().take(k).repeat()`. You should use `dataset.take(k).cache().repeat()` inste
ad.
```




```
2024-05-30 09:02:41.847356: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

Now we bild a model with two *GRU* layers.

In [ ]:
```
multi_step_model = tf.keras.models.Sequential()
multi_step_model.add(tf.keras.layers.GRU(32,
                                          return_sequences=True,
                                          input_shape=x_train_multi.shape[-2:]))
multi_step_model.add(tf.keras.layers.GRU(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), loss
multi_step_model.summary()
x_train_multi.shape[-2:]
```

Model: "sequential_11"

Layer (type)	Output Shape	Param #
gru_14 (GRU)	(None, 20, 32)	3,552
gru_15 (GRU)	(None, 16)	2,400
dense_11 (Dense)	(None, 10)	170

Total params: 6,122 (23.91 KB)

Trainable params: 6,122 (23.91 KB)

Non-trainable params: 0 (0.00 B)

Out[ ]: (20, 3)

As before, we observe that the number of parameter is smaller compared to the model with the LSTM layers, due to the different architectures.

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

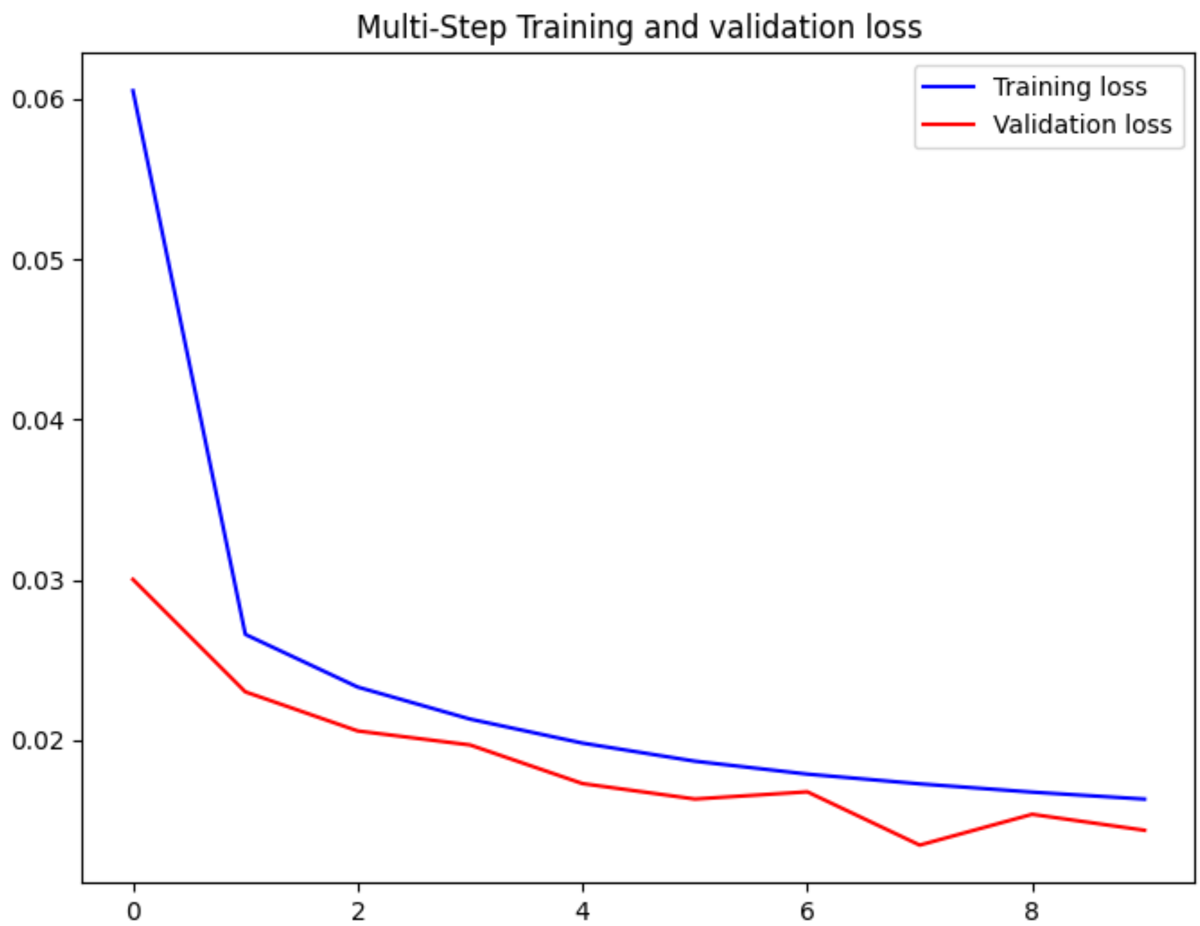
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
(256, 10)
2024-05-30 09:02:42.115515: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

The training time is longer for this more complex model.

In [ ]:
```
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 ━━━━━━━━━━━━━━━━━━━━ 22s 10ms/step - loss: 0.1428 - val_loss: 0.0300
Epoch 2/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - loss: 0.0277 - val_loss: 0.0230
Epoch 3/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - loss: 0.0239 - val_loss: 0.0206
Epoch 4/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 21s 10ms/step - loss: 0.0217 - val_loss: 0.0197
Epoch 5/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - loss: 0.0202 - val_loss: 0.0173
Epoch 6/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - loss: 0.0190 - val_loss: 0.0163
Epoch 7/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - loss: 0.0181 - val_loss: 0.0168
Epoch 8/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 21s 10ms/step - loss: 0.0174 - val_loss: 0.0135
Epoch 9/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 21s 10ms/step - loss: 0.0169 - val_loss: 0.0154
Epoch 10/10
2000/2000 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - loss: 0.0164 - val_loss: 0.0144
```

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

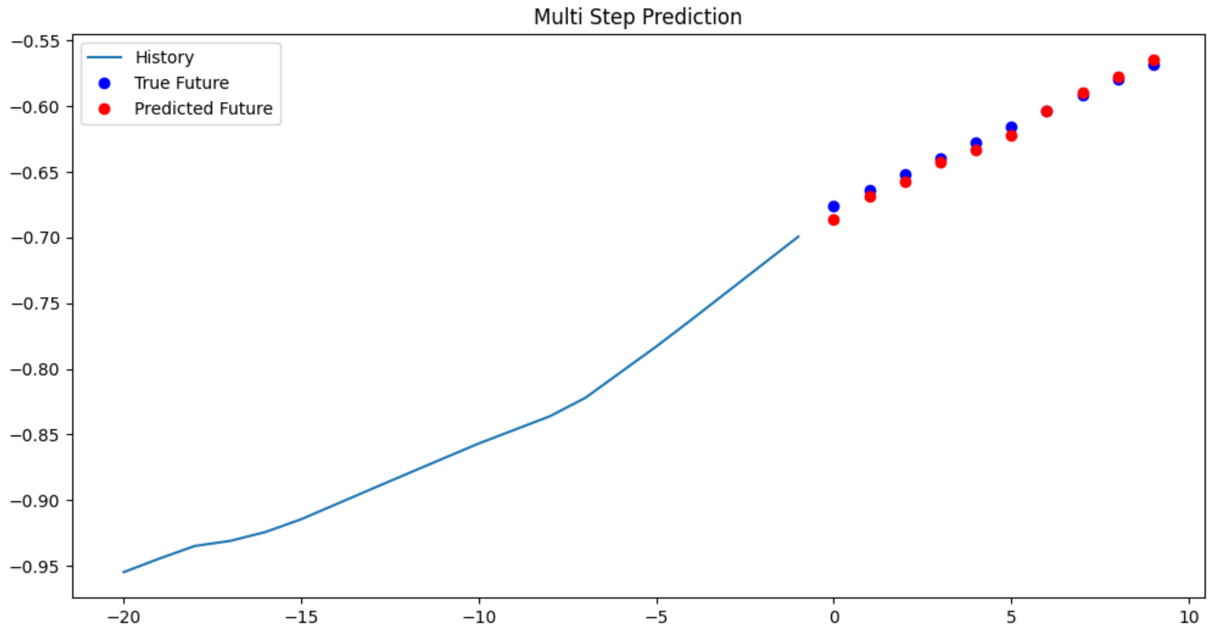


Here, we see once again a big decrease in training loss at the very beginning. The validation loss tend to shrink next to the training loss, which could indicate that the resulting network will not suffer from overfitting. Moreover, it is possible that the training needs some more epochs in order to let the loss converge at a certain value. However, the situation here looks quite better (regarding to that the model already reached to optimum) compared to the previous model.

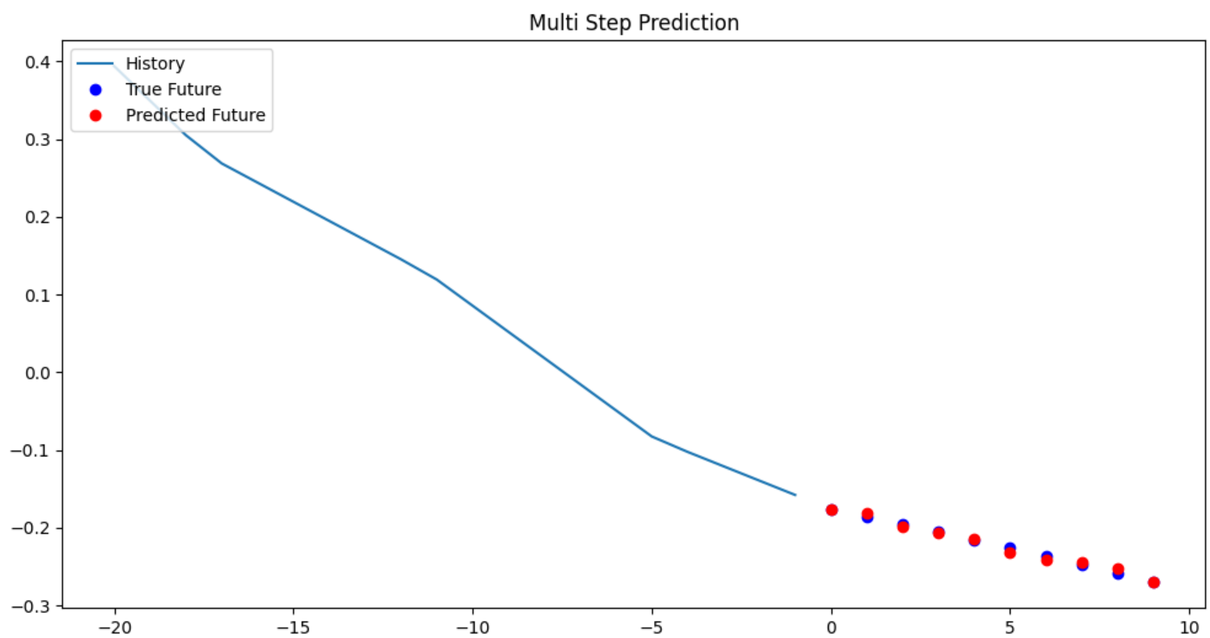
According to time, the training was slower compared to the other models. This is due to the more complex architecture which come with more parameters and , thus, a higher computational effort. Anyway, once again it was a little bit fast than the training of the model from the original notebook, which could be due to the architecture coming with fewer parameters but also to the different machine on that the training was ran.

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

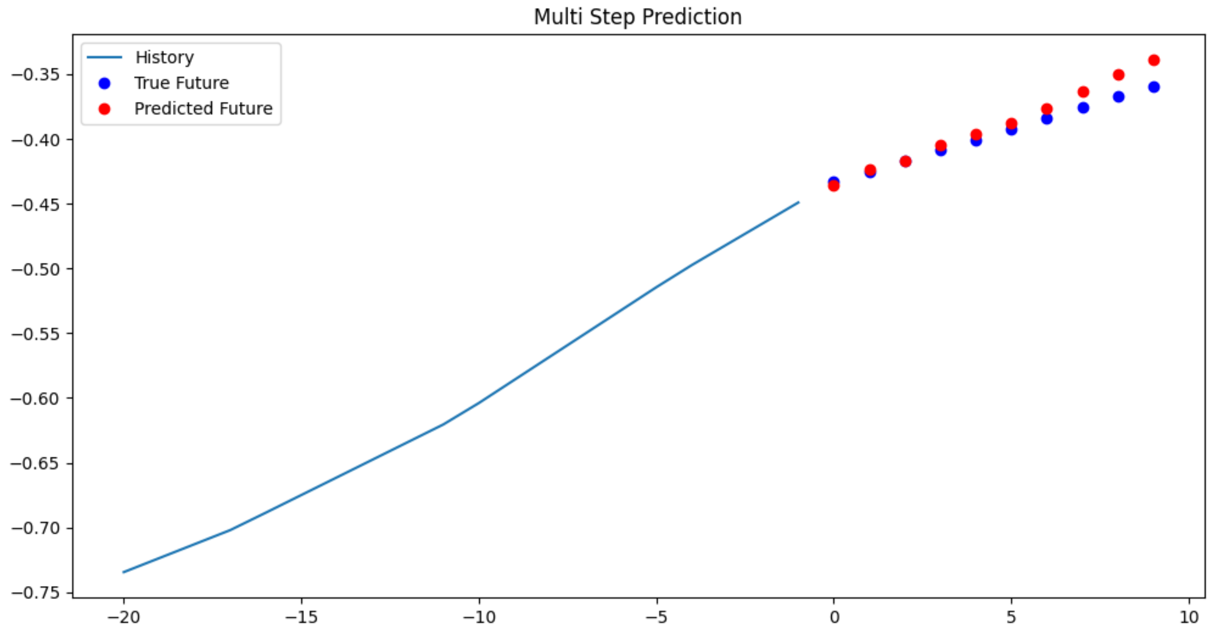
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



8/8 ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step



8/8 ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step



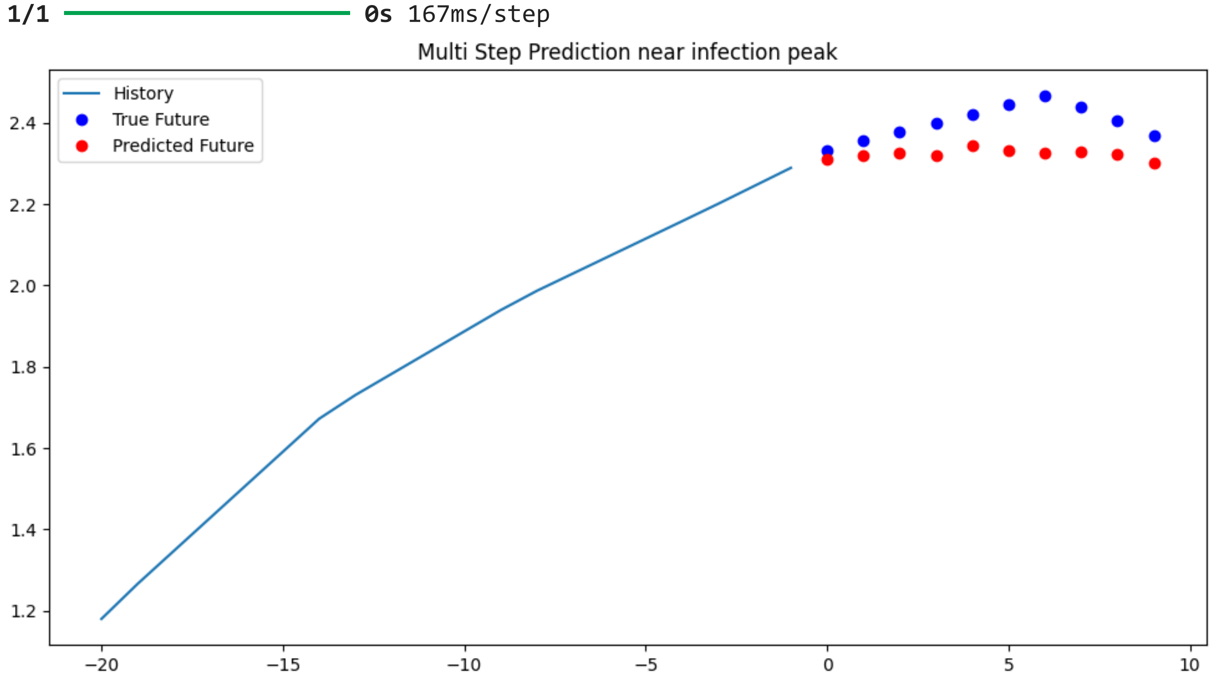

```
2024-05-30 09:06:08.637510: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

Looking at the predictions they are quite good. In some cases, one sees that the difference between true value and prediction is larger when forcasting is made for a more distant future (third example).

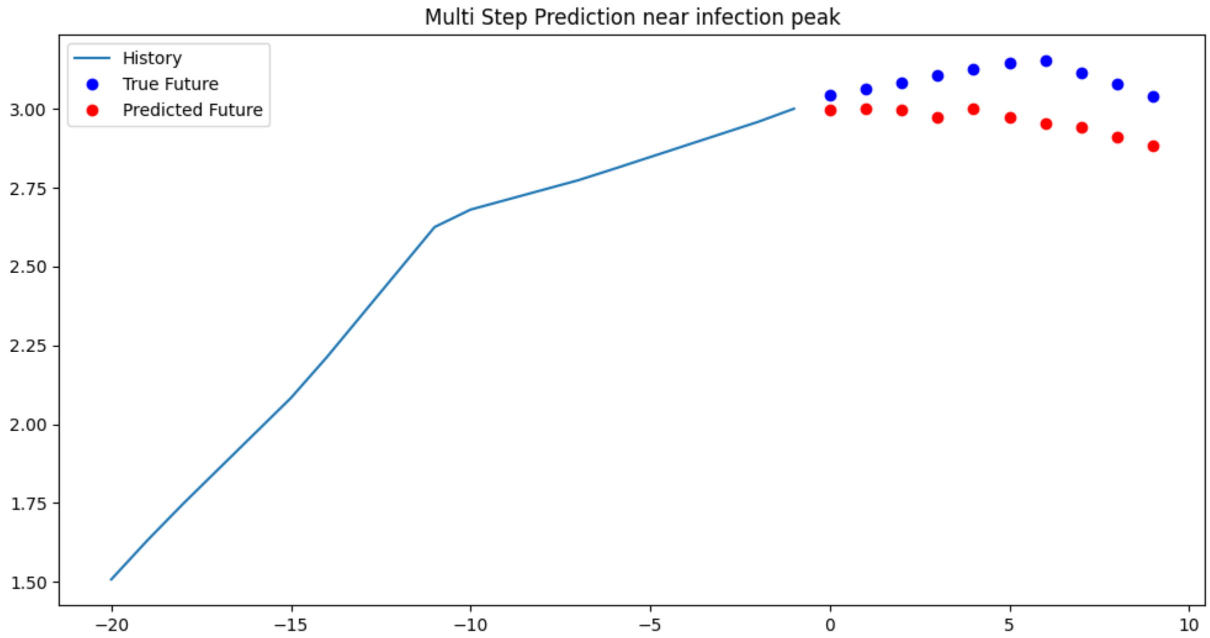
Now, let's look at the prediction near infection peak.

In [ ]:
```
for x, y in peak_data_multi.take(3):
    multi_step_plot(x[0], y[0], multi_step_model.predict(x)[0], title="Multi Step P
```

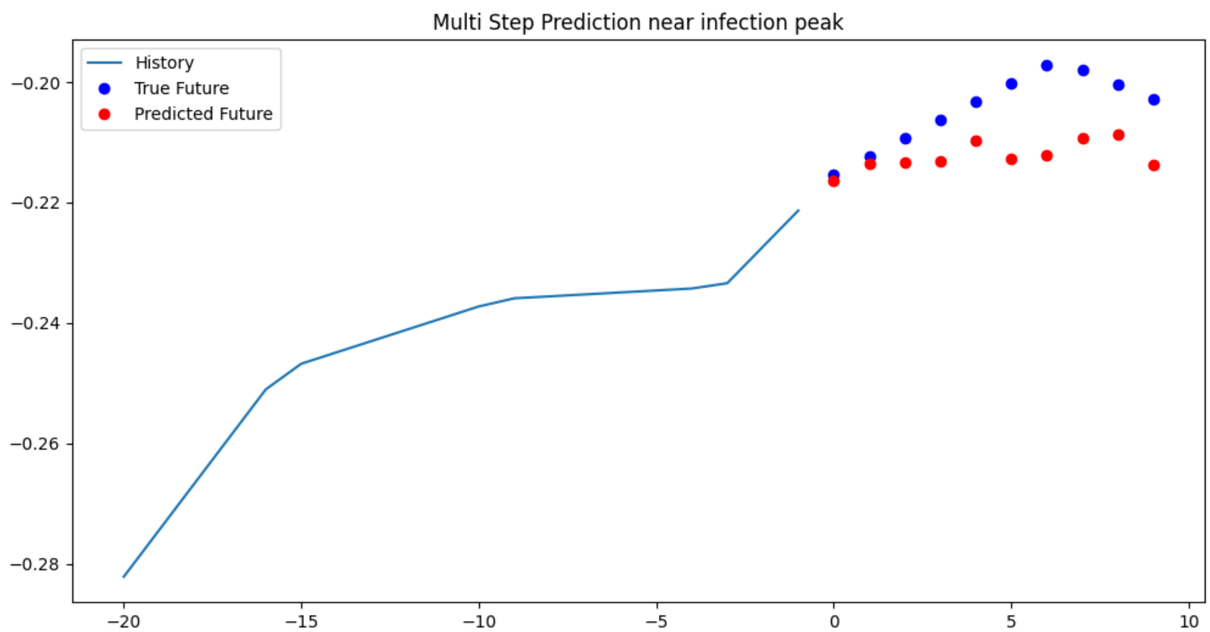
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```




```
2024-05-30 09:06:09.171313: W tensorflow/core/framework/local_rendezvous.cc:404] Loc
al rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

Here, we see that the model is likely to miss the peak and predicts a mor flat trajectory. So once again, it seems to have trouble with the change in the direction of the trajectory, i.e. from increasing infections to decreasing infections. This could be due to the training data. As we have seen in the plotted examples, there are some timelines that are likely to haven't reached the peak yet, and, therefore, were increasing all the time. So maby, one could try to focus the training on timelines showing a global within the, e.g., first 300 days, in order to ensure a complete infection cycle - if those data exists.