lstm_multivariate_to_multivariate

May 29, 2023

1 MULTIVARIATE PREDICTION USING MULTIVARIATE FEATURES WITH LSTM SEQ2SEQ

1.1 HERE WE USED: 12h for running the running inference over the next 1h

[1]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from keras.models import Sequential, save_model, load_model
from keras.layers import Bidirectional, LSTM, Dropout, Dense
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import mean_absolute_percentage_error
import os
import time
from tensorflow.keras.callbacks import CSVLogger, EarlyStopping
from tensorflow.keras.layers import BatchNormalization, ConvLSTM2D, RepeatVector
from keras.layers.core import Dense, Dropout, Activation, Flatten, Reshape
from tensorflow.keras.layers import TimeDistributed

import tensorflow as tf
# physical_devices = tf.config.list_physical_devices('GPU')
# tf.config.experimental.set_memory_growth(physical_devices[0], enable=True)

models_path = "../saved_models/normal/may2023"
# read dataset may2023
df = pd.read_pickle("../../data/20230319_RTU_Dataset_PPC-Lab/combined_may2023.
 ↪pkl")
```

```
2023-05-29 17:57:30.670432: I tensorflow/core/util/port.cc:110] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-05-29 17:57:30.699390: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
```

```
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

[2]:

```
# Data Loader Parameters
BATCH_SIZE = 128
BUFFER_SIZE = 100

# LSTM Parameters
EVALUATION_INTERVAL = 200
PATIENCE = 10
PAST_WINDOW_SIZE = 144                  # ----------- 12H
FUTURE_WINDOW_SIZE = 12                 # ----------- 1H
STEP = 3

# Reproducibility
SEED = 13
tf.random.set_seed(SEED)
```

[22]:

```
def create_sequence(dataset, target, window, future):
    x_sequence, y_sequence = [], []
    for index in range(len(dataset) - window - future):
        x_sequence.append(dataset[index: index + window])
        y_sequence.append(target[index + window: index + window + future])
    return (np.asarray(x_sequence), np.asarray(y_sequence))

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()

def multivariate_multioutput_data(dataset, target, start_index, end_index,␣
 ↪history_size, target_size, step):
    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)
        data.append(dataset[indices])
        labels.append(target[i:i+target_size])

    return np.array(data)[:,:,:,np.newaxis,np.newaxis], np.array(labels)[:,:,:
 ↪,np.newaxis,np.newaxis]

# for x, y in val_data_multi.take(10):
#     multi_step_output_plot(np.squeeze(x[0]), np.squeeze(y[0]), np.
 ↪squeeze(model.predict(x[0][np.newaxis,:,:,:,:])), df_original_scale)
def multi_step_output_plot(history, true_future, prediction, dataset):
    plt.figure(figsize=(18, 6))
    num_in = create_time_steps(len(history))
    num_out = len(true_future)
    evaluate = []

    for i, (var, c) in enumerate(zip(dataset.columns, ['b','r', 'g'])):
        plt.plot(num_in, np.array(history[:, i]), c, label=var)
        plt.plot(np.arange(num_out)/STEP, np.array(true_future[:,i]), c+'o',␣
 ↪markersize=5, alpha=0.5, label=f"True {var.title()}")
        if prediction.any():
            plt.plot(np.arange(num_out)/STEP, np.array(prediction[:,i]), '*',␣
 ↪markersize=5, alpha=0.5, label=f"Predicted {var.title()}")

    plt.legend(loc='upper left')
    plt.show()
    return evaluate

def create_time_steps(length):
    return list(range(-length, 0))

def evaluate_predictions(predictions_seq, y_test_seq):
    MSE = []
    MAPE = mean_absolute_percentage_error(predictions_seq, y_test_seq)
    for pred in range(len(predictions_seq)):
        mse = mean_squared_error(y_test_seq[pred], predictions_seq[pred])
        MSE.append(mse)

    mean_mse = sum(MSE)/len(MSE)

    return mean_mse, MAPE

def find_max_error(predictions, y_test, mean_mse, std_mse):
    max_errors = 0
```

```python
    for pred in range(len(y_test)):
        mse = mean_squared_error(y_test[pred], predictions[pred])
    if mse > mean_mse + std_mse:
        max_errors += 1
    return max_errors

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def build_model_simplified(input_timesteps, output_timesteps, num_links, num_inputs):
    model = Sequential()
    model.add(BatchNormalization(name = 'batch_norm_0', input_shape = (input_timesteps, num_inputs, 1, 1)))
    model.add(ConvLSTM2D(name ='conv_lstm_1',
                         filters = 64, kernel_size = (10, 1),
                         padding = 'same',
                         return_sequences = False))

    model.add(Dropout(0.30, name = 'dropout_1'))
    model.add(BatchNormalization(name = 'batch_norm_1'))

#     model.add(ConvLSTM2D(name ='conv_lstm_2',
#                          filters = 64, kernel_size = (5, 1),
#                          padding='same',
#                          return_sequences = False))

#     model.add(Dropout(0.20, name = 'dropout_2'))
#     model.add(BatchNormalization(name = 'batch_norm_2'))

    model.add(Flatten())
    model.add(RepeatVector(output_timesteps))
    model.add(Reshape((output_timesteps, num_inputs, 1, 64)))

#     model.add(ConvLSTM2D(name ='conv_lstm_3',
#                          filters = 64, kernel_size = (10, 1),
#                          padding='same',
#                          return_sequences = True))

#     model.add(Dropout(0.20, name = 'dropout_3'))
#     model.add(BatchNormalization(name = 'batch_norm_3'))

    model.add(ConvLSTM2D(name ='conv_lstm_4',
                         filters = 64, kernel_size = (5, 1),
                         padding='same',
                         return_sequences = True))
    model.add(TimeDistributed(Dense(units=1, name = 'dense_1', activation =␣
 ↪'relu')))
    model.add(Dense(units=1, name = 'dense_2'))

#     optimizer = RMSprop() #lr=0.0001, rho=0.9, epsilon=1e-08, decay=0.9)
#     optimizer = tf.keras.optimizers.Adam(0.1)
    optimizer = tf.keras.optimizers.RMSprop(lr=0.003, clipvalue=1.0)
    model.compile(loss = "mse", optimizer = optimizer, metrics = ['mae', 'mse'])
    return model

def build_model(input_timesteps, output_timesteps, num_links, num_inputs):
    # COPY PASTA
    # https://github.com/niklascp/bus-arrival-convlstm/blob/master/jupyter/
 ↪ConvLSTM_3x15min_10x64-5x64-10x64-5x64-Comparison.ipynb

    model = Sequential()
    model.add(BatchNormalization(name = 'batch_norm_0', input_shape =␣
 ↪(input_timesteps, num_inputs, 1, 1)))
    model.add(ConvLSTM2D(name ='conv_lstm_1',
                         filters = 64, kernel_size = (10, 1),
                         padding = 'same',
                         return_sequences = True))

    model.add(Dropout(0.30, name = 'dropout_1'))
    model.add(BatchNormalization(name = 'batch_norm_1'))

    model.add(ConvLSTM2D(name ='conv_lstm_2',
                         filters = 64, kernel_size = (5, 1),
                         padding='same',
                         return_sequences = False))

    model.add(Dropout(0.20, name = 'dropout_2'))
    model.add(BatchNormalization(name = 'batch_norm_2'))

    model.add(Flatten())
    model.add(RepeatVector(output_timesteps))
    model.add(Reshape((output_timesteps, num_inputs, 1, 64)))

    model.add(ConvLSTM2D(name ='conv_lstm_3',
                         filters = 64, kernel_size = (10, 1),
                         padding='same',
                         return_sequences = True))

    model.add(Dropout(0.20, name = 'dropout_3'))
    model.add(BatchNormalization(name = 'batch_norm_3'))
    model.add(ConvLSTM2D(name ='conv_lstm_4',
                         filters = 64, kernel_size = (5, 1),
                         padding='same',
                         return_sequences = True))

    model.add(TimeDistributed(Dense(units=1, name = 'dense_1', activation =␣
 ↪'relu')))
    model.add(Dense(units=1, name = 'dense_2', activation = 'linear'))

#     optimizer = RMSprop() #lr=0.0001, rho=0.9, epsilon=1e-08, decay=0.9)
#     optimizer = tf.keras.optimizers.Adam(0.1)
    optimizer = tf.keras.optimizers.RMSprop(lr=0.004, clipvalue=1.0)
    model.compile(loss = "mse", optimizer = optimizer, metrics = ['mae', 'mse'])
    return model

def my_mean_absolute_percentage_error(y_true, y_pred):
    error = 0
    for i in range(len(y_true)):
        if y_true[i] != 0:
            error += abs((y_true[i] - y_pred[i]) / y_true[i])

    mape = (error / len(y_true)) * 100
    return mape
```

[4]:

```
# Normalizing the values
standard_scaler = preprocessing.StandardScaler()
print(df.head())
scaled_df = standard_scaler.fit_transform(df[['MEM_USAGE', 'CPU_USAGE',␣
 ↪'TEMP']])
print(scaled_df[:10])

training_size = int(len(scaled_df) * 0.8)

print('Scaled_df shape: ' + str(scaled_df.shape))
print('Size of the dataset: %d' % (len(scaled_df)))
print('Size of training: %d' % (training_size))
```

```
   MEM_USAGE  CPU_USAGE     PS1_V    TEMP
0  35.555417  27.343750  5.435294  28.687
1  35.555417   6.367041  5.435294  28.687
2  35.555417   7.142857  5.435294  28.687
3  35.555417  27.306273  5.435294  28.687
4  35.555417   5.639098  5.435294  28.687
[[ 0.48139574  1.13540371  0.74576055]
 [ 0.48139574 -0.66263387  0.74576055]
 [ 0.48139574 -0.59613411  0.74576055]
 [ 0.48139574  1.13219134  0.74576055]
 [ 0.48139574 -0.7250302   0.74576055]
 [ 0.48139574 -0.73742406  0.74576055]
 [ 0.48139574 -0.6369512   0.74576055]
 [ 0.48139574 -0.91168167  0.74576055]
 [ 0.48139574 -0.63268673  0.74576055]
 [ 0.48139574 -0.27087286  0.74576055]]
Scaled_df shape: (3733, 3)
Size of the dataset: 3733
Size of training: 2986
```

[5]:

```
x_train_multi, y_train_multi = multivariate_multioutput_data(scaled_df,␣
 ↪scaled_df, 0,
                                                 training_size,␣
 ↪PAST_WINDOW_SIZE,
                                                 FUTURE_WINDOW_SIZE, STEP)
x_val_multi, y_val_multi = multivariate_multioutput_data(scaled_df, scaled_df,
                                             training_size, None,␣
 ↪PAST_WINDOW_SIZE,
                                             FUTURE_WINDOW_SIZE, STEP)
```

[6]:

```
x_train_multi.shape
```

[6]:

```
(2842, 48, 3, 1, 1)
```

[7]:

```
y_train_multi.shape
```

[7]:

```
(2842, 12, 3, 1, 1)
```

[8]:

```
x_val_multi.shape
```

[8]:

```
(591, 48, 3, 1, 1)
```

[9]:

```
y_val_multi.shape
```

[9]:

```
(591, 12, 3, 1, 1)
```

[10]:

```
BATCH_SIZE = 128

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()
```

```
2023-05-29 17:57:33.250826: I
```

```
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.259598: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.259910: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.262270: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.262731: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.263011: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.616988: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.617110: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.617181: I
```

```
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2023-05-29 17:57:33.617249: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 14115 MB memory:  -> device:
0, name: NVIDIA RTX A5000 Laptop GPU, pci bus id: 0000:01:00.0, compute
capability: 8.6
```

[ ]:

```
EPOCHS = 30
steps_per_epoch = 350
validation_steps = 500

modelstart = time.time()
early_stopping = EarlyStopping(monitor='val_loss', patience = PATIENCE,␣
 ↪restore_best_weights=True)
model = build_model(x_train_multi.shape[1], FUTURE_WINDOW_SIZE, y_train_multi.
 ↪shape[2], x_train_multi.shape[2])
print(model.summary())

# Train
print("\nTRAIN MODEL...")
history = model.fit(train_data_multi,
                    epochs = EPOCHS,
                    validation_data=val_data_multi,
                    steps_per_epoch=steps_per_epoch,
                    validation_steps=validation_steps,
                    verbose=1,
                    callbacks=[early_stopping])
model.save('multi-output-timesteps.h5')
print("\nModel Runtime: %0.2f Minutes"%((time.time() - modelstart)/60))
```

[ ]:

```
plot_train_history(history, 'Multi-Step, Multi-Output Training and validation␣
 ↪loss')
```

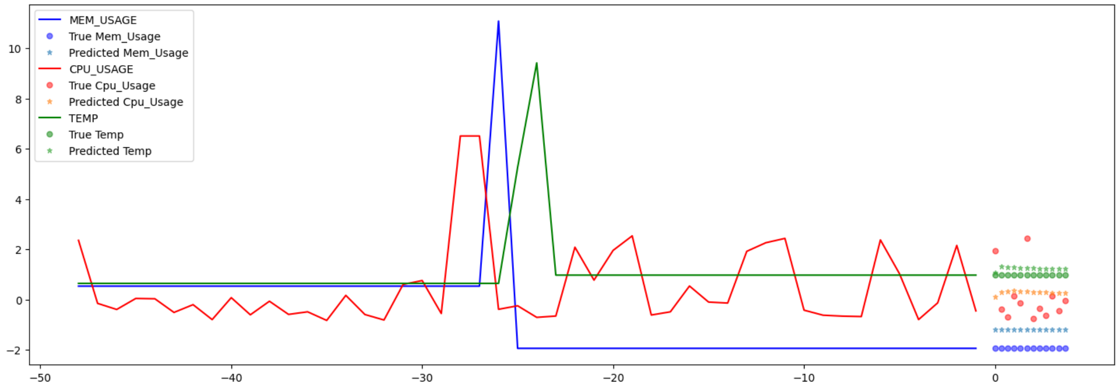
[12]:

```
from tensorflow import keras
model = keras.models.load_model("./best_multi_multi_lstm.h5")

column_names = ['MEM_USAGE', 'CPU_USAGE', 'TEMP']
df_original_scale = pd.DataFrame(data=scaled_df, columns=column_names)

for x, y in val_data_multi.take(10):
    multi_step_output_plot(np.squeeze(x[0]), np.squeeze(y[0]), np.squeeze(model.
 ↪predict(x[0][np.newaxis,:,:,:,:])), df_original_scale)
```

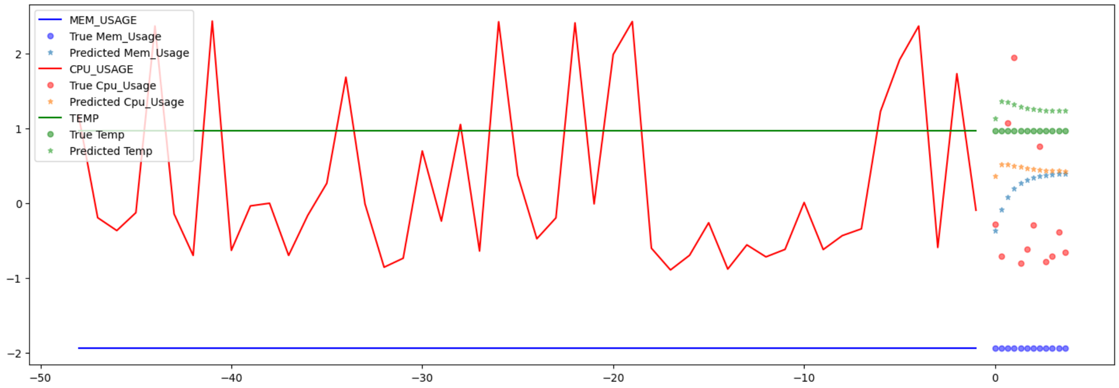
```
2023-05-29 17:57:44.405271: I tensorflow/core/common_runtime/executor.cc:1197]
```

```
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'Placeholder/_1' with dtype double and shape
[591,12,3,1,1]
         [[{{node Placeholder/_1}}]]
```

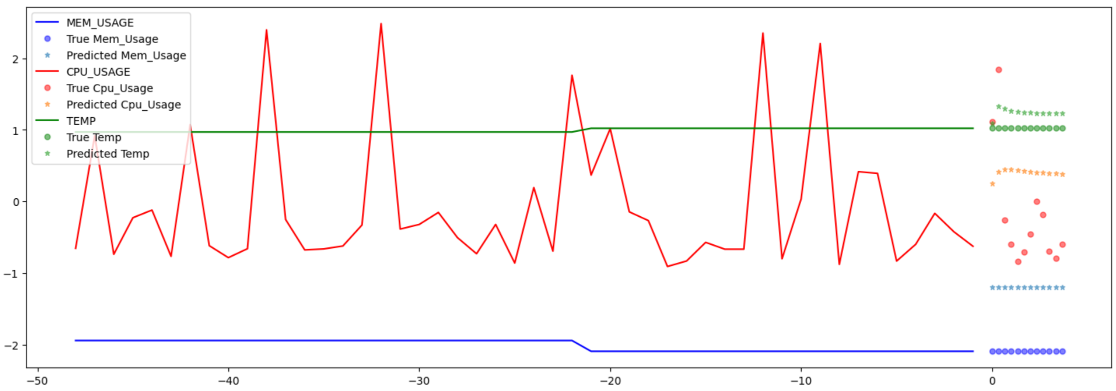
```
1/1 [==============================] - 1s 579ms/step
```



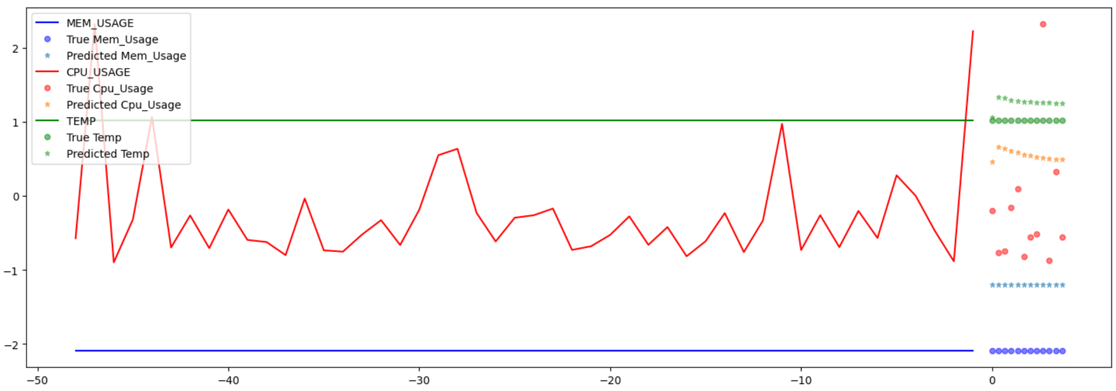
```
1/1 [==============================] - 0s 49ms/step
```



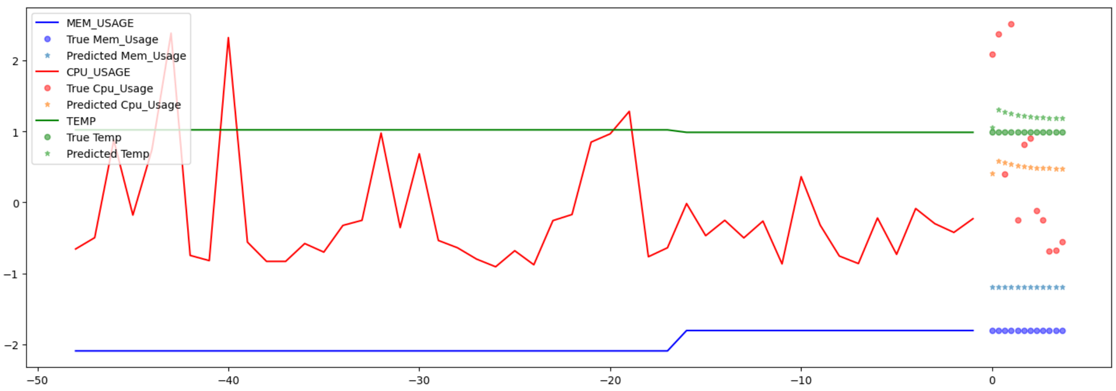
```
1/1 [==============================] - 0s 49ms/step
```



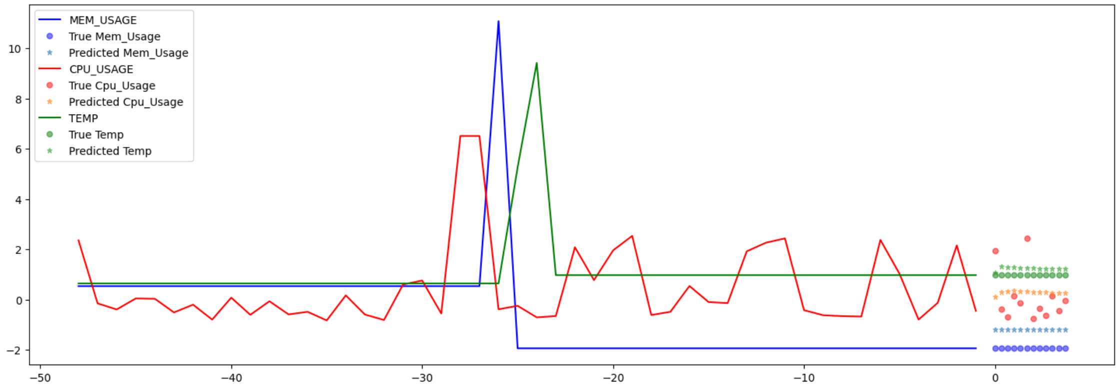
```
1/1 [==============================] - 0s 49ms/step
```



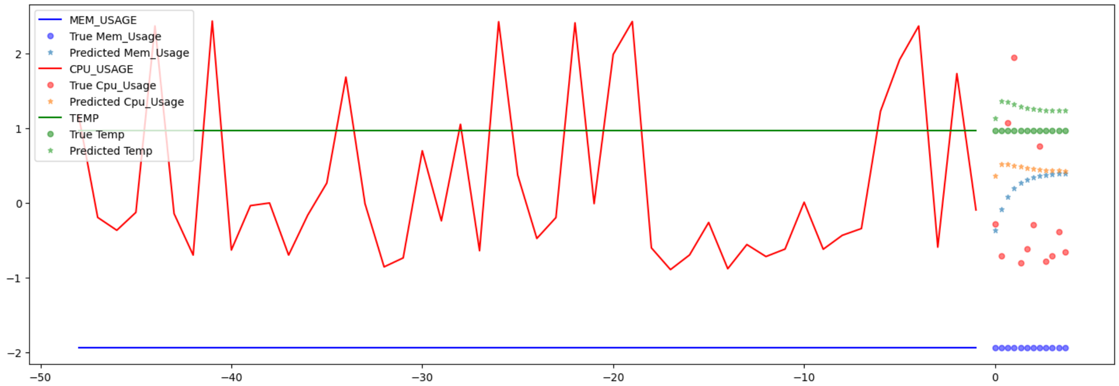
```
1/1 [==============================] - 0s 50ms/step
```



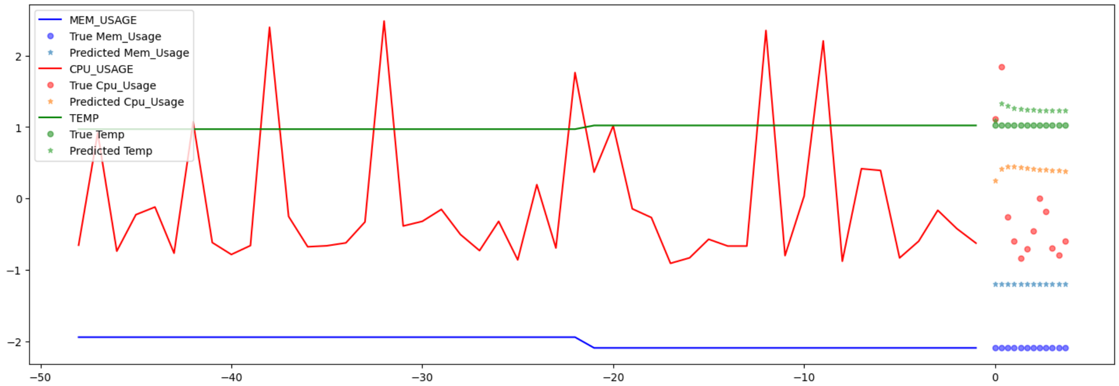
```
1/1 [==============================] - 0s 48ms/step
```



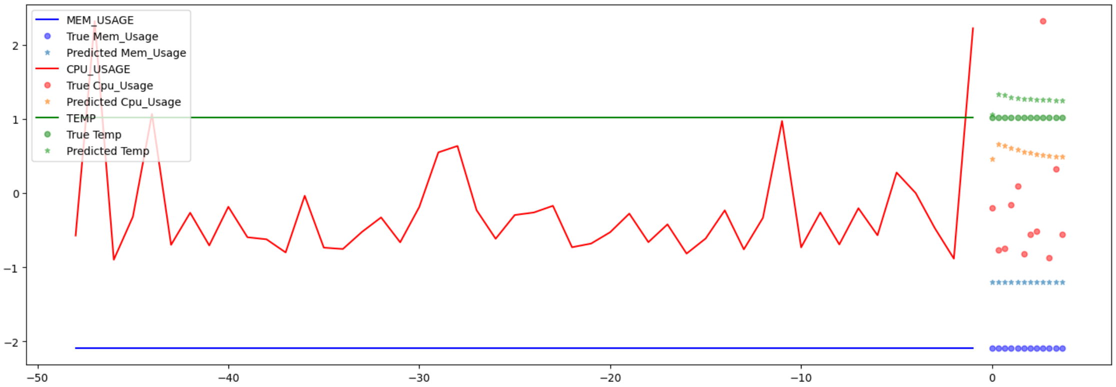
```
1/1 [==============================] - 0s 48ms/step
```



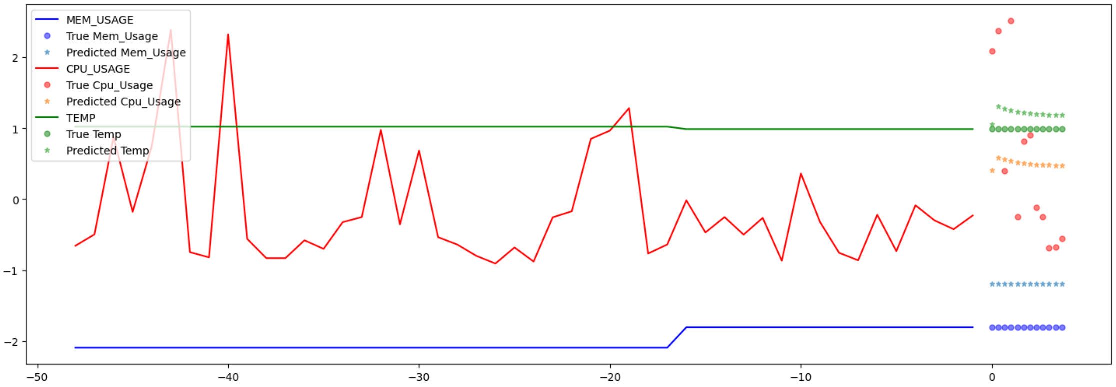
```
1/1 [==============================] - 0s 48ms/step
```



```
1/1 [==============================] - 0s 58ms/step
```



```
1/1 [==============================] - 0s 48ms/step
```



2 EVALUATE PERFORMANCE

[13]:
```
import tensorflow as tf
train_data_multi = tf.data.Dataset.from_tensor_slices((x_train_multi,␣
  ↪y_train_multi))
train_data_multi = train_data_multi.cache().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)

train_data_multi
```

[13]:
```
<_BatchDataset element_spec=(TensorSpec(shape=(None, 48, 3, 1, 1),
dtype=tf.float64, name=None), TensorSpec(shape=(None, 12, 3, 1, 1),
dtype=tf.float64, name=None))>
```

[14]:

```
history_mems = []
history_cpus = []
history_temps = []

predicted_mems = []
predicted_cpus = []
predicted_temps = []

original_mems = []
original_cpus = []
original_temps = []

for x, y in train_data_multi:
    hx_mem = np.squeeze(x[0])[:,0]
    hx_cpu = np.squeeze(x[0])[:,1]
    hx_temp = np.squeeze(x[0])[:,2]

    history_mems.append(hx_mem)
    history_cpus.append(hx_cpu)
    history_temps.append(hx_temp)


for x, y in val_data_multi:

    prediction = np.squeeze(model.predict(x[0][np.newaxis,:,:,:,:], verbose =␣
 ↪0))
    pred_mems = prediction[:,0]
    pred_cpus = prediction[:,1]
    pred_temps = prediction[:,2]
    ori_mems = np.squeeze(y[0])[:,0]
    ori_cpus = np.squeeze(y[0])[:,1]
    ori_temps = np.squeeze(y[0])[:,2]

    predicted_mems.append(pred_mems)
    original_mems.append(ori_mems)

    predicted_cpus.append(pred_cpus)
    original_cpus.append(ori_cpus)

    predicted_temps.append(pred_temps)
    original_temps.append(ori_temps)
```

```
2023-05-29 17:57:50.833659: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
```

```
for placeholder tensor 'Placeholder/_1' with dtype double and shape
[2842,12,3,1,1]
         [[{{node Placeholder/_1}}]]
2023-05-29 17:57:50.879246: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'Placeholder/_1' with dtype double and shape
[591,12,3,1,1]
         [[{{node Placeholder/_1}}]]
```

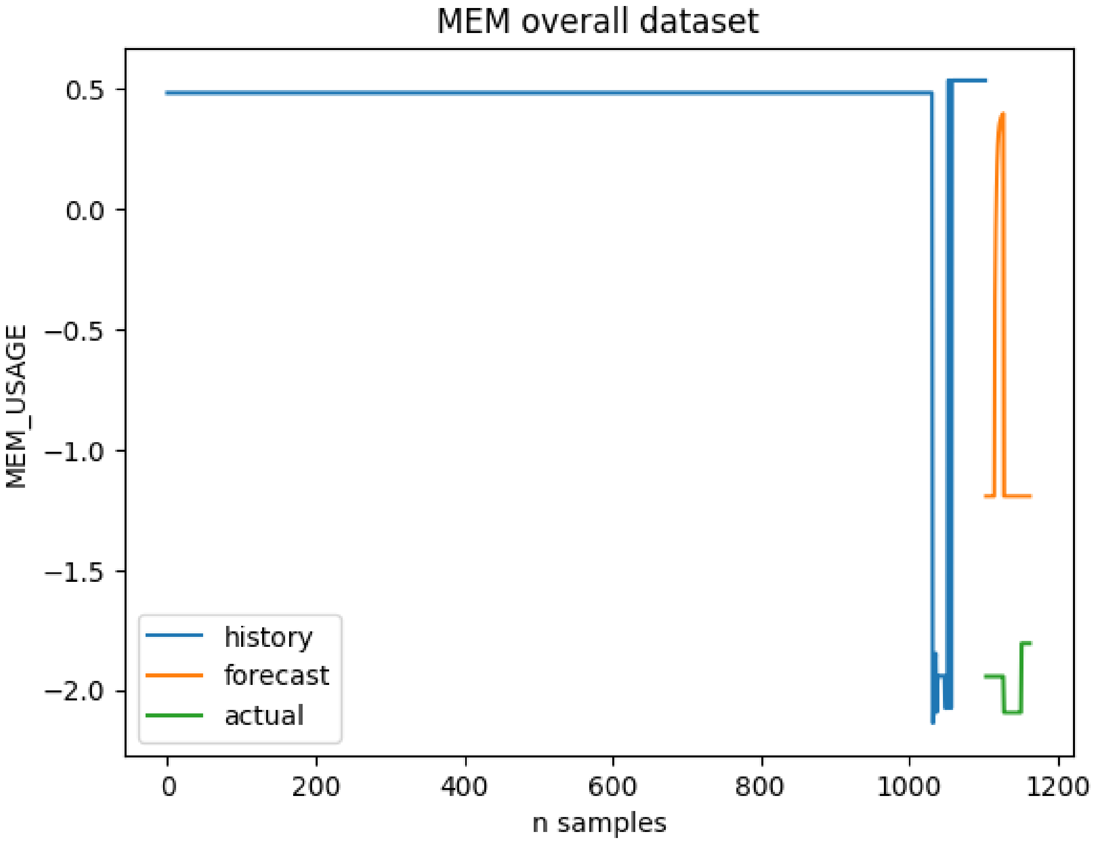
[15]:

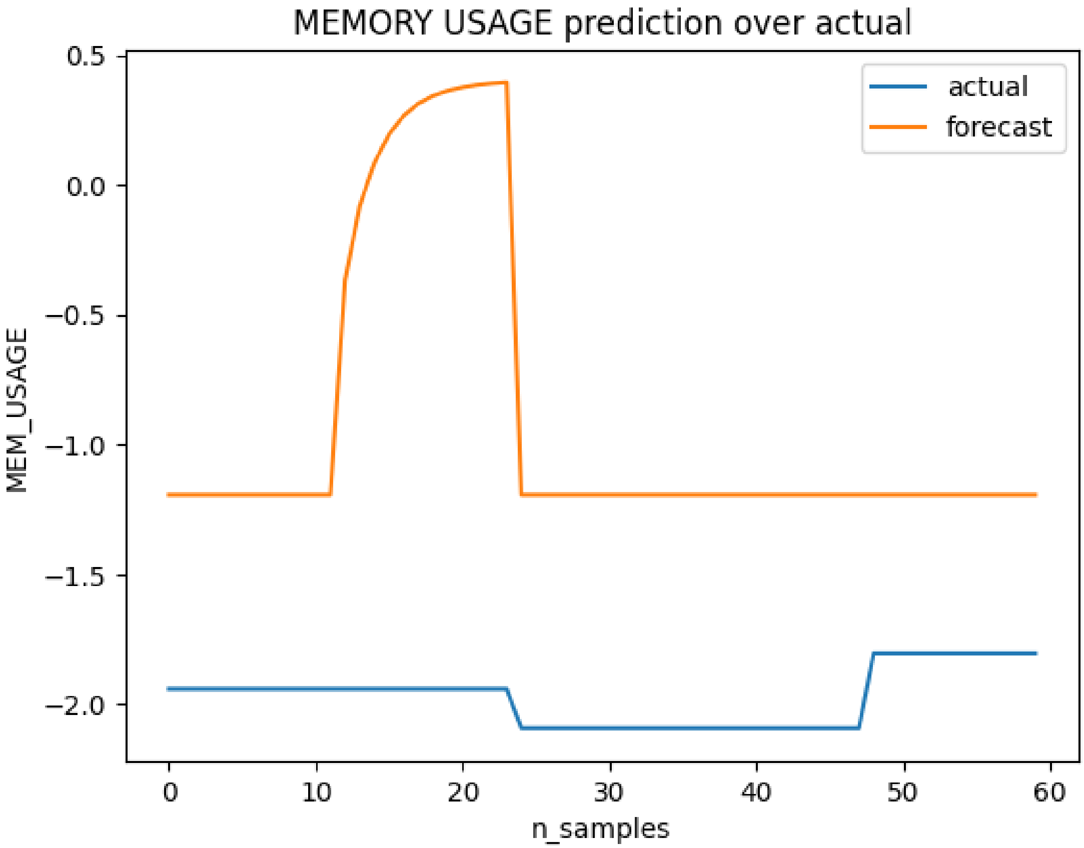
```
history_mem_usage = np.concatenate(history_mems, axis=0)
history_cpu_usage = np.concatenate(history_cpus, axis=0)
history_temperatures = np.concatenate(history_temps, axis=0)

predicted_mem_usage = np.concatenate(predicted_mems, axis=0)
predicted_cpu_usage = np.concatenate(predicted_cpus, axis=0)
predicted_temperatures = np.concatenate(predicted_temps, axis=0)

original_mem_usage = np.concatenate(original_mems, axis=0)
original_cpu_usage = np.concatenate(original_cpus, axis=0)
original_temperatures = np.concatenate(original_temps, axis=0)
```

2.1 Error for MEMORY USAGE

[16]:

```
plt.title("MEM overall dataset")
plt.xlabel("n samples")
plt.ylabel("MEM_USAGE")
plt.plot(range(len(history_mem_usage)),history_mem_usage, label="history")
plt.plot(range(len(history_mem_usage), len(history_mem_usage) +␣
 ↪len(predicted_mem_usage)),predicted_mem_usage, label="forecast")
plt.plot(range(len(history_mem_usage), len(history_mem_usage) +␣
 ↪len(predicted_mem_usage)),original_mem_usage, label="actual")

plt.legend()
plt.show()


x = range(len(predicted_mem_usage))
plt.title("MEMORY USAGE prediction over actual")
plt.xlabel("n_samples")
plt.ylabel("MEM_USAGE")
plt.plot(x,original_mem_usage, label="actual")
plt.plot(x,predicted_mem_usage, label="forecast")
plt.legend()
plt.show()
```


MEM overall dataset
MEM_USAGE
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
200
400
600
800
1000
1200
n samples
history
forecast
actual



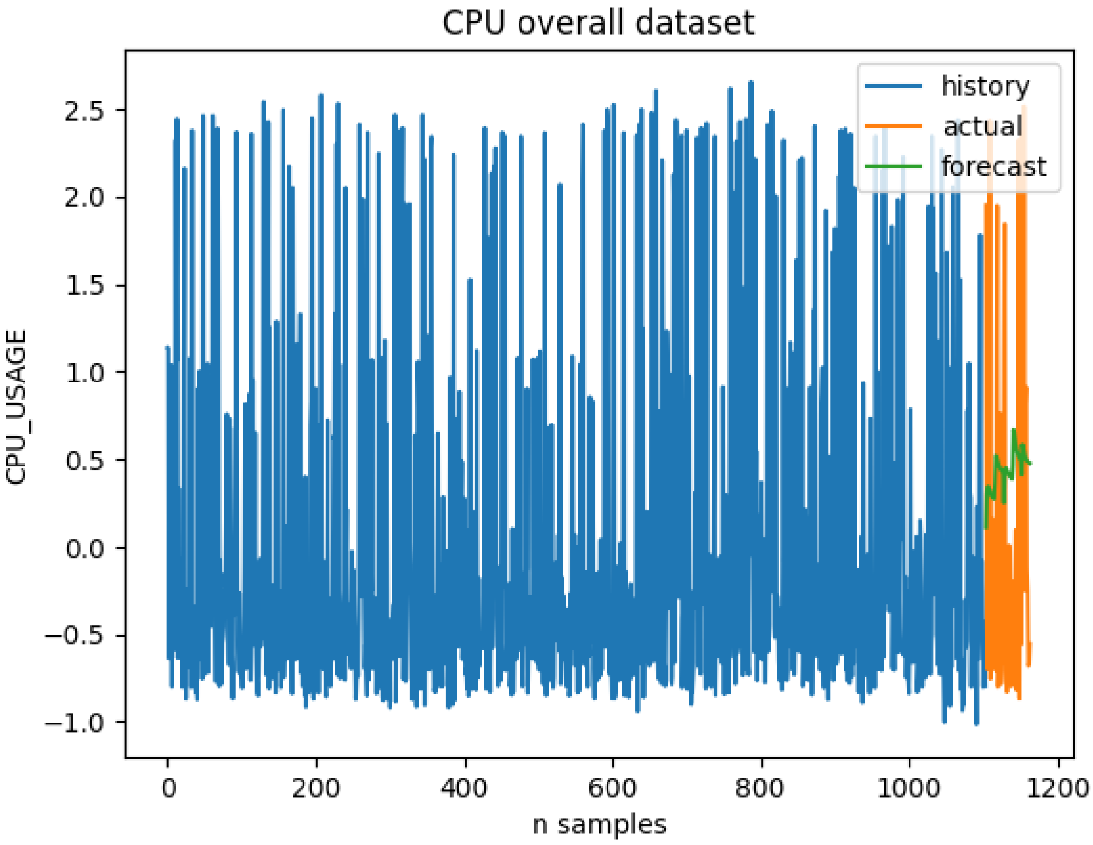
[23]:
```
my_mean_absolute_percentage_error(original_mem_usage, predicted_mem_usage)
```

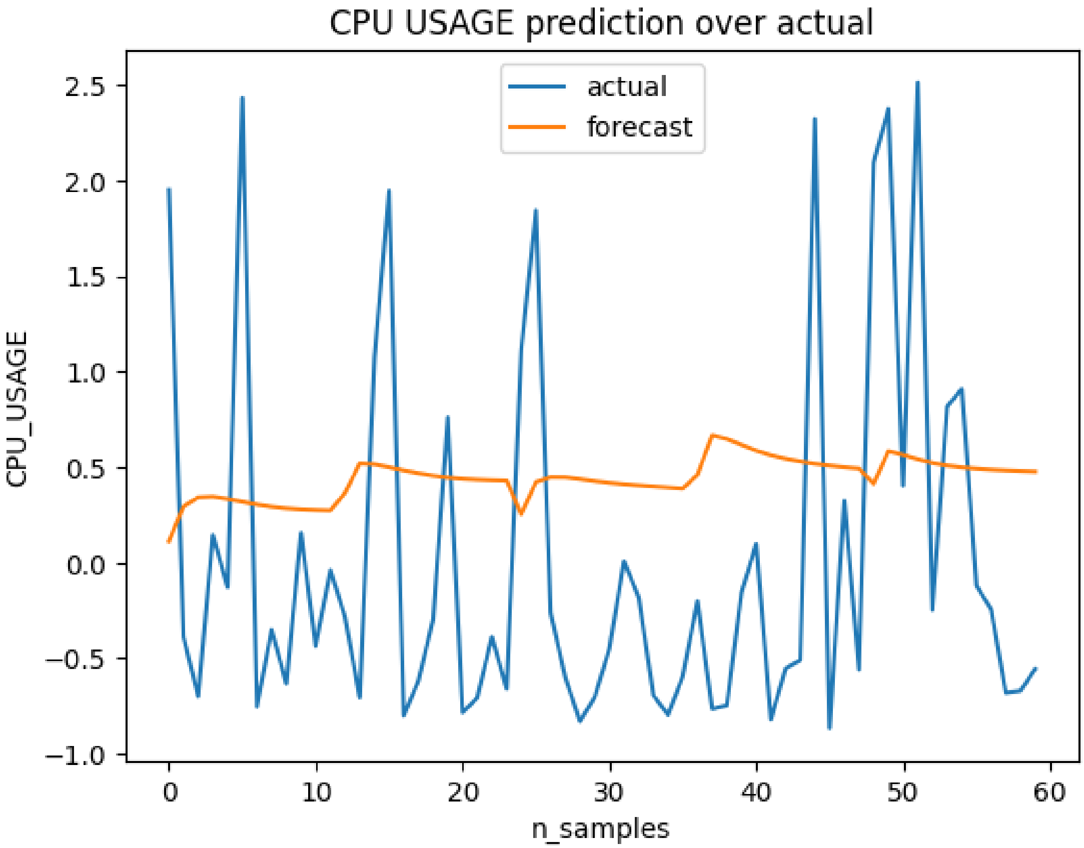
[23]:
```
53.93274616330089
```

2.2 Error for CPU USAGE

[19]:
```
plt.title("CPU overall dataset")
plt.xlabel("n samples")
plt.ylabel("CPU_USAGE")
plt.plot(range(len(history_cpu_usage)),history_cpu_usage, label="history")
plt.plot(range(len(history_cpu_usage), len(history_cpu_usage) +␣
 ↪len(predicted_cpu_usage)),original_cpu_usage, label="actual")
plt.plot(range(len(history_cpu_usage), len(history_cpu_usage) +␣
 ↪len(predicted_cpu_usage)),predicted_cpu_usage, label="forecast")

plt.legend()
plt.show()

x = range(len(predicted_mem_usage))
```

```
plt.title("CPU USAGE prediction over actual")
plt.xlabel("n_samples")
plt.ylabel("CPU_USAGE")
plt.plot(x,original_cpu_usage, label="actual")
plt.plot(x,predicted_cpu_usage, label="forecast")
plt.legend()
plt.show()
```





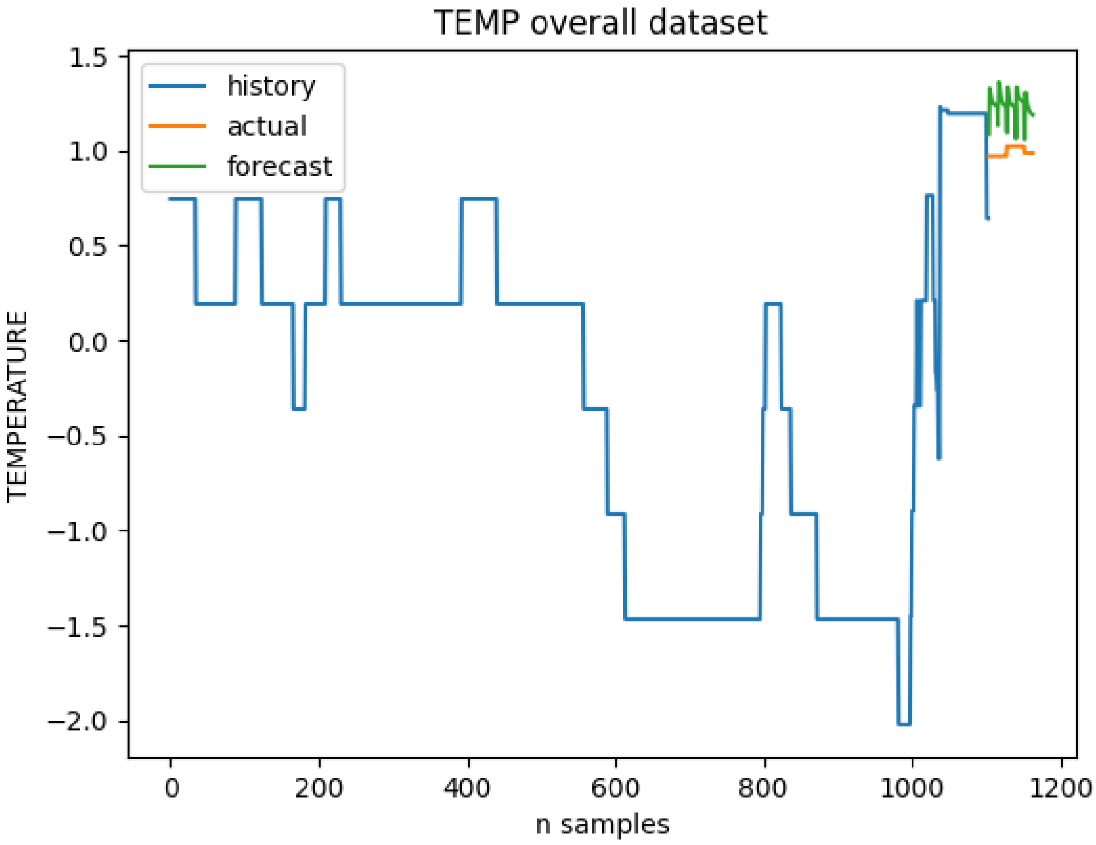
[24]:
```
my_mean_absolute_percentage_error(original_cpu_usage, predicted_cpu_usage)
```

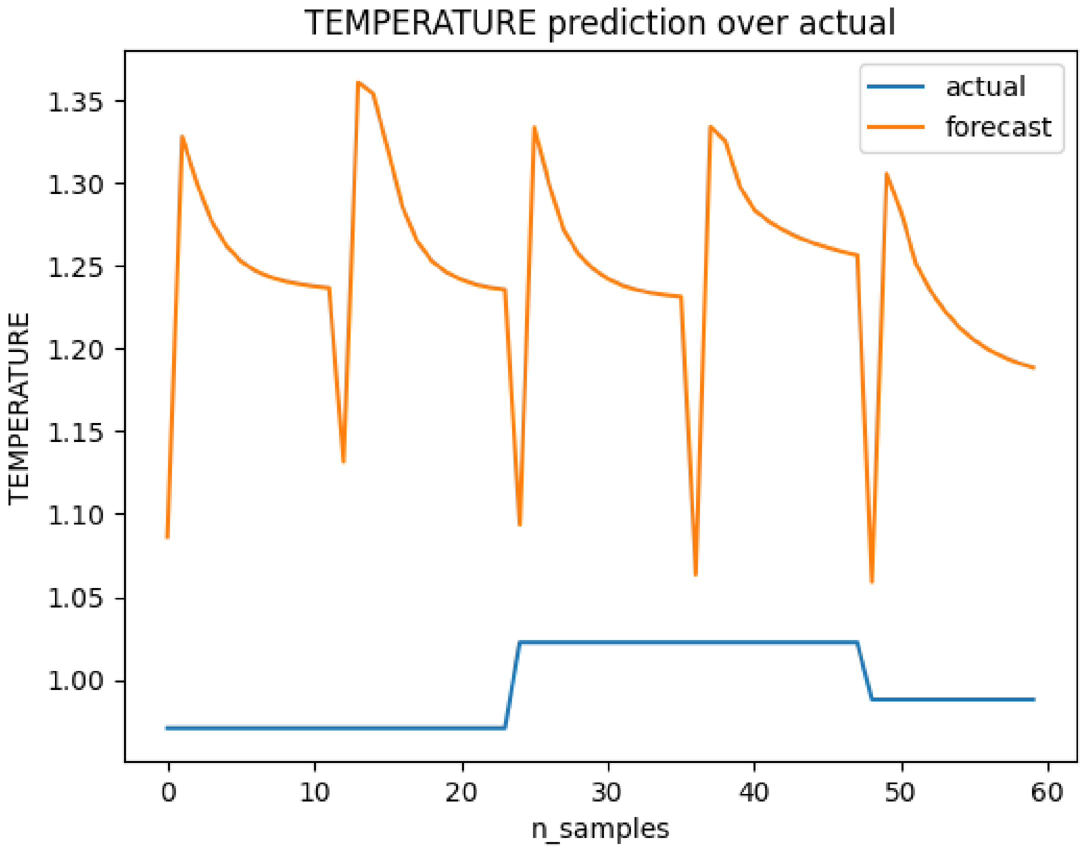
[24]:
```
283.49445102609775
```

2.3 Error for temperature

[133]:
```
plt.title("TEMP overall dataset")
plt.xlabel("n samples")
plt.ylabel("TEMPERATURE")
plt.plot(range(len(history_temperatures)),history_temperatures, label="history")
plt.plot(range(len(history_temperatures), len(history_temperatures) +␣
 ↪len(predicted_temperatures)),original_temperatures, label="actual")
plt.plot(range(len(history_temperatures), len(history_temperatures) +␣
 ↪len(predicted_temperatures)),predicted_temperatures, label="forecast")

plt.legend()
plt.show()

x = range(len(predicted_mem_usage))
```

```
plt.title("TEMPERATURE prediction over actual")
plt.xlabel("n_samples")
plt.ylabel("TEMPERATURE")
plt.plot(x,original_temperatures, label="actual")
plt.plot(x,predicted_temperatures, label="forecast")
plt.legend()
plt.show()
```





```
[25]: my_mean_absolute_percentage_error(original_temperatures, predicted_temperatures)
```

```
[25]: 25.2723204810813
```

Overall the only metrics which seems to reach a reasonable accuracy is the temperature with 75% of accuracy (1-MAPE), but the model lacks any understanding of the other metrics, I dont consider this a reliable result