Assignment 3: Time-Series Data **Qiao Wang**

By using various methods, the test results in terms of MAE are shown in the table below. Due to computational limitations, the largest Epoch number I can use is about 20, some models can only run as little as 2 Epochs. The best result I can get is 2.43. The model is a stacked GRU model with 32 units in each layer, 0.6 of dropout and 15 Epochs. It seems using a combination of 1D_convnets and LSTM doesn't have very good results. Also stacked LSTM is not as good as stacked GRU models in my setting.

Model	Number of Units in Each Layer	LSTM/GRU	Dropout	Epochs	Test MAE
Combination of 1D_Convnets and RNN1	16	LSTM	0.4/0.6	5	2.69
Combination of 1D_Convnets and RNN2	32	LSTM	0.25/0.5	5	2.95
Stacked GRU1	16	GRU	0.4	3	2.91
Stacked GRU2	16	GRU	0.5	2	2.61
Stacked GRU3	64	GRU	0.5	15	2.5
Stacked GRU4	32	GRU	0.5	15	2.44
Stacked GRU5	32	GRU	0.6	15	2.43
Stacked LSTM1	16	LSTM	0.25/0.6	10	2.49
Stacked LSTM2	48	LSTM	0.3/0.6	10	2.49
Stacked LSTM3	64	LSTM	0.25/0.5	6	2.8
Stacked LSTM4	128	LSTM	0.25/0.6	2	2.49
Stacked LSTM5	32	LSTM	0.4/0.6	20	2.62
Stacked LSTM6	64	LSTM	0.25/0.6	6	2.45

The best model is shown below, please refer other models in Github.

Deep learning for timeseries

A temperature-forecasting example

In [1]:

```
#!wget https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
#!unzip jena_climate_2009_2016.csv.zip
```

Inspecting the data of the Jena weather dataset

In [2]:

```
import os
fname = os.path.join("jena_climate_2009_2016.csv")

with open(fname) as f:
    data = f.read()

lines = data.split("\n")
header = lines[0].split(",")
lines = lines[1:]
print(header)
print(len(lines))
```

```
['"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)"']
420451
```

Parsing the data

In [3]:

```
import numpy as np
temperature = np.zeros((len(lines),))
raw_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(",")[1:]]
    temperature[i] = values[1]
    raw_data[i, :] = values[:]
```

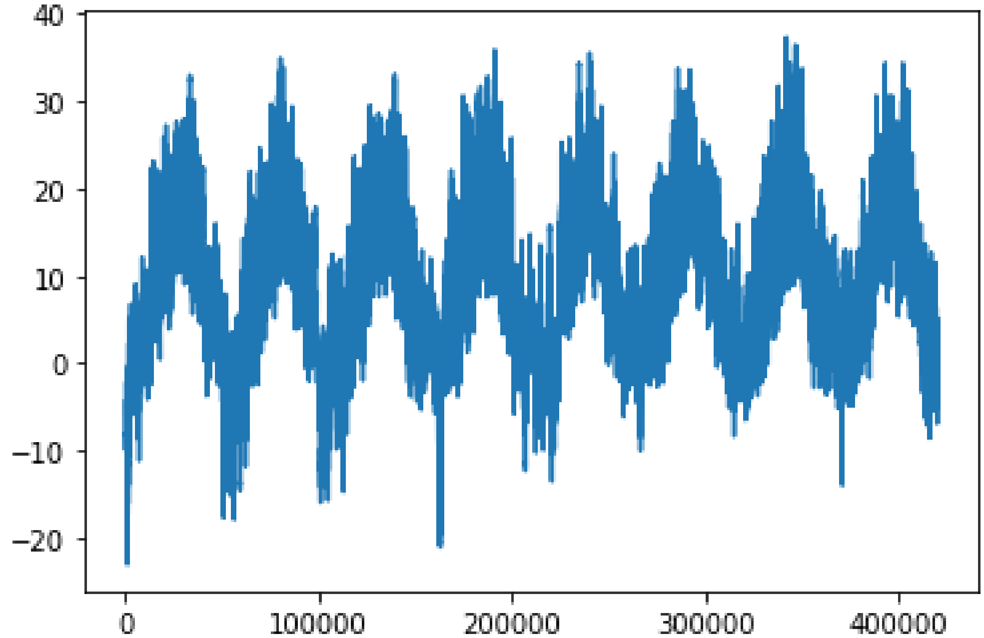
Plotting the temperature timeseries

In [4]:

```
from matplotlib import pyplot as plt
plt.plot(range(len(temperature)), temperature)
```

Out[4]:

```
[<matplotlib.lines.Line2D at 0x7f71e61668d0>]
```



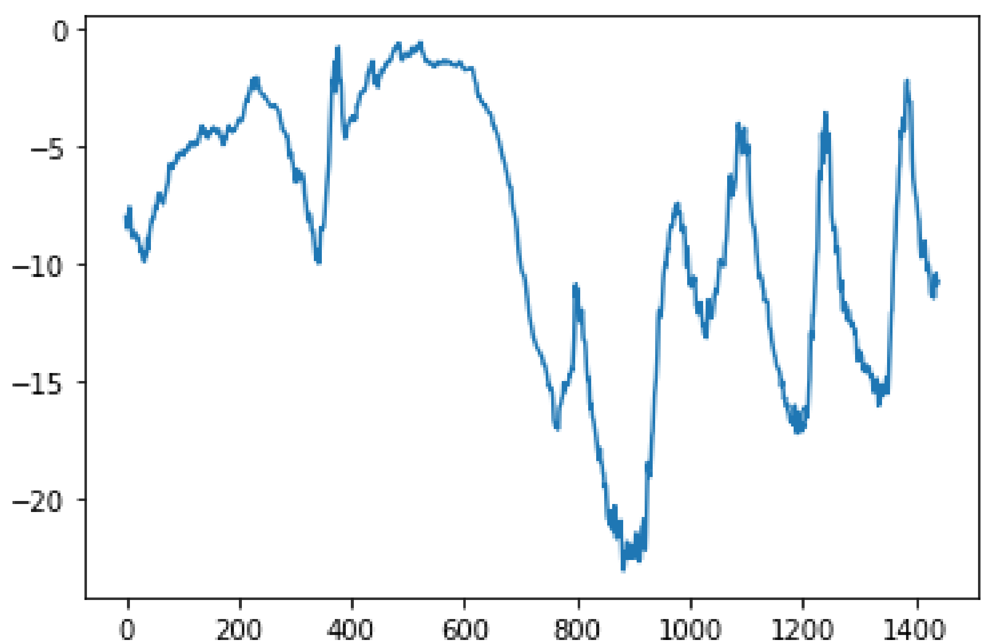
Plotting the first 10 days of the temperature timeseries

In [5]:

```
plt.plot(range(1440), temperature[:1440])
```

Out[5]:

```
[<matplotlib.lines.Line2D at 0x7f71de066208>]
```



Computing the number of samples we'll use for each data split

In [6]:

```
num_train_samples = int(0.5 * len(raw_data))
num_val_samples = int(0.25 * len(raw_data))
num_test_samples = len(raw_data) - num_train_samples - num_val_samples
```

```
print("num_train_samples:", num_train_samples)
print("num_val_samples:", num_val_samples)
print("num_test_samples:", num_test_samples)
```

```
num_train_samples: 210225
num_val_samples: 105112
num_test_samples: 105114
```

Preparing the data

Normalizing the data

In [7]:

```
mean = raw_data[:num_train_samples].mean(axis=0)
raw_data -= mean
std = raw_data[:num_train_samples].std(axis=0)
raw_data /= std
```

In [8]:

```
import numpy as np
from tensorflow import keras
int_sequence = np.arange(10)
dummy_dataset = keras.utils.timeseries_dataset_from_array(
    data=int_sequence[:-3],
    targets=int_sequence[3:],
    sequence_length=3,
    batch_size=2,
)

for inputs, targets in dummy_dataset:
    for i in range(inputs.shape[0]):
        print([int(x) for x in inputs[i]], int(targets[i]))
```

```
[0, 1, 2] 3
[1, 2, 3] 4
[2, 3, 4] 5
[3, 4, 5] 6
[4, 5, 6] 7
```

Instantiating datasets for training, validation, and testing

In [9]:

```
sampling_rate = 6
sequence_length = 120
delay = sampling_rate * (sequence_length + 24 - 1)
batch_size = 256

train_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=0,
```

```
    end_index=num_train_samples)

val_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples,
    end_index=num_train_samples + num_val_samples)

test_dataset = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples + num_val_samples)
```

Inspecting the output of one of our datasets

In [10]:

```
for samples, targets in train_dataset:
    print("samples shape:", samples.shape)
    print("targets shape:", targets.shape)
    break
```

```
samples shape: (256, 120, 14)
targets shape: (256,)
```

Stacking recurrent layers

Training and evaluating a dropout-regularized, stacked GRU model

In [11]:

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.6, return_sequences=True)(inputs)
x = layers.GRU(32, recurrent_dropout=0.6)(x)
x = layers.Dropout(0.6)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_gru_dropout141.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=15,
```

```
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_stacked_gru_dropout141.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/15
819/819 [==============================] - 216s 260ms/step - loss: 25.8742 -
mae: 3.7903 - val_loss: 9.7699 - val_mae: 2.4145
Epoch 2/15
819/819 [==============================] - 200s 244ms/step - loss: 16.0230 -
mae: 3.0885 - val_loss: 8.8086 - val_mae: 2.2995
Epoch 3/15
819/819 [==============================] - 200s 244ms/step - loss: 15.1241 -
mae: 2.9995 - val_loss: 8.8301 - val_mae: 2.2993
Epoch 4/15
819/819 [==============================] - 199s 243ms/step - loss: 14.4743 -
mae: 2.9360 - val_loss: 8.8495 - val_mae: 2.3030
Epoch 5/15
819/819 [==============================] - 199s 243ms/step - loss: 13.8034 -
mae: 2.8698 - val_loss: 8.6986 - val_mae: 2.2799
Epoch 6/15
819/819 [==============================] - 195s 238ms/step - loss: 13.3083 -
mae: 2.8191 - val_loss: 9.0716 - val_mae: 2.3463
Epoch 7/15
819/819 [==============================] - 198s 241ms/step - loss: 12.7921 -
mae: 2.7649 - val_loss: 8.7279 - val_mae: 2.2918
Epoch 8/15
819/819 [==============================] - 197s 241ms/step - loss: 12.4516 -
mae: 2.7284 - val_loss: 8.9426 - val_mae: 2.3201
Epoch 9/15
819/819 [==============================] - 199s 242ms/step - loss: 12.1117 -
mae: 2.6922 - val_loss: 9.0313 - val_mae: 2.3363
Epoch 10/15
819/819 [==============================] - 197s 241ms/step - loss: 11.8556 -
mae: 2.6665 - val_loss: 9.1256 - val_mae: 2.3430
Epoch 11/15
819/819 [==============================] - 198s 242ms/step - loss: 11.6512 -
mae: 2.6419 - val_loss: 8.8285 - val_mae: 2.3017
Epoch 12/15
819/819 [==============================] - 199s 243ms/step - loss: 11.3220 -
mae: 2.6066 - val_loss: 9.0036 - val_mae: 2.3303
Epoch 13/15
819/819 [==============================] - 199s 243ms/step - loss: 11.1801 -
mae: 2.5907 - val_loss: 9.1073 - val_mae: 2.3444
Epoch 14/15
819/819 [==============================] - 198s 242ms/step - loss: 10.9890 -
mae: 2.5700 - val_loss: 8.9941 - val_mae: 2.3306
Epoch 15/15
819/819 [==============================] - 196s 240ms/step - loss: 10.8157 -
mae: 2.5506 - val_loss: 9.0226 - val_mae: 2.3351
405/405 [==============================] - 20s 49ms/step - loss: 9.6833 - ma
e: 2.4278
Test MAE: 2.43
```

Summary