An Introduction to Deep Learning With Python

[6.4] A temperature-forecasting problem

Prof. Yuzo Iano

pgs: 207 - 220

Inspecting the data of the Jena weather dataset

The dataset was download from https://www.kaggle.com/stytch16/jena-climate-2009-2016/version/1 (https://www.kaggle.com/stytch16/jena-climate-2009-2016/version/1)

In [1]:
```
import os

data_dir = 'jena_climate'
fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')
f = open(fname)
data = f.read()
f.close()

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)"']
420551
```

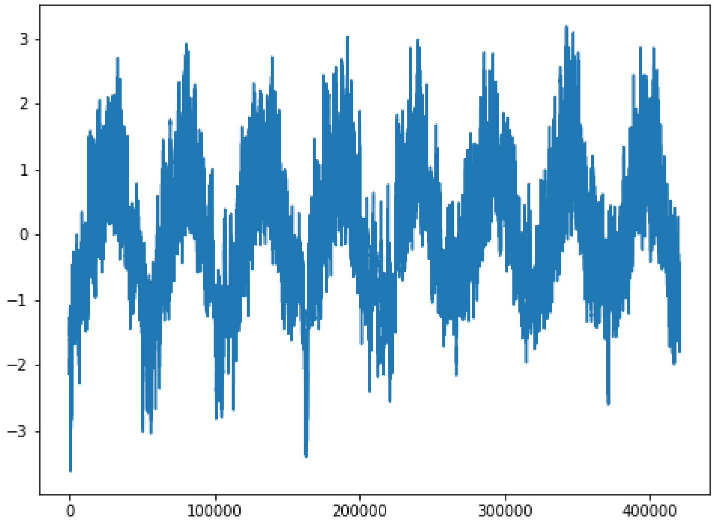
Parsing the data

In [2]:
```
import numpy as np

float_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(',')[1:]]
    float_data[i, :] = values
```

Plotting the temperature timeseries

In [21]:
```
import matplotlib.pyplot as plt

temp = float_data[:, 1]
plt.figure(figsize=(8, 6))
plt.plot(range(len(temp)), temp)
```

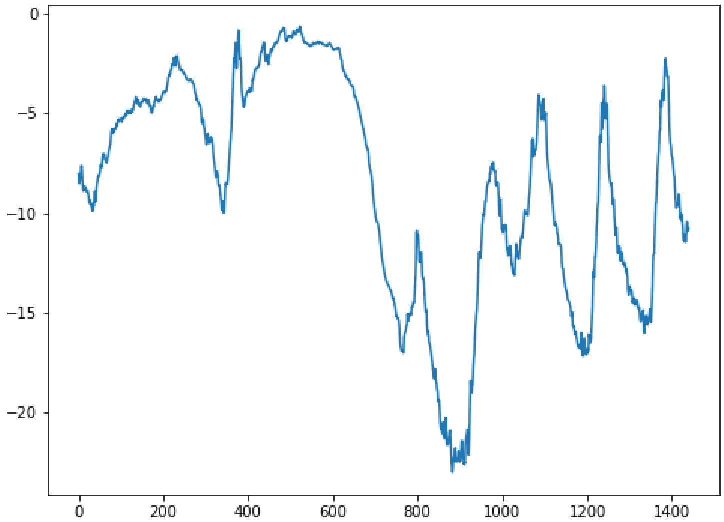
Out[21]: [<matplotlib.lines.Line2D at 0x14e80c1db38>]



Plotting the first 10 days of the temperature timeseries

In [4]:
```
plt.figure(figsize=(8, 6))
plt.plot(range(1440), temp[:1440])
```

Out[4]: [<matplotlib.lines.Line2D at 0x14ee73cab00>]



Normalizing the data

In [5]:
```
mean = float_data[:200000].mean(axis=0)
float_data -= mean
std = float_data[:200000].std(axis=0)
float_data /= std
```

Generator yielding timseries samples and their targets

In [6]:
```
def generator(data, lookback, delay, min_index, max_index,
              shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)

        samples = np.zeros((len(rows),
                           lookback // step,
                           data.shape[-1]))
        targets = np.zeros((len(rows),))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices]
            targets[j] = data[rows[j] + delay][1]
        yield samples, targets
```

Preparing the training, validation, and test generators

In [7]:
```
lookback = 1440
step = 6
delay = 144
batch_size = 128

train_gen = generator(float_data,
                      lookback=lookback,
                      delay=delay,
                      min_index=0,
                      max_index=200000,
                      shuffle=True,
                      step=step,
                      batch_size=batch_size)
val_gen = generator(float_data,
                    lookback=lookback,
                    delay=delay,
                    min_index=200001,
                    max_index=300000,
                    step=step,
                    batch_size=batch_size)
test_gen = generator(float_data,
                     lookback=lookback,
                     delay=delay,
                     min_index=300001,
                     max_index=None,
                     step=step,
                     batch_size=batch_size)

val_steps = (300000 - 200001 - lookback) // batch_size


test_steps = (len(float_data) - 300001 - lookback) // batch_size
```

Computing the common-sense baseline MAE

In [8]:
```
def evaluate_naive_method():
    batch_maes = []
    for step in range(val_steps):
        samples, targets = next(val_gen)
        preds = samples[:, -1, 1]
        mae = np.mean(np.abs(preds - targets))
        batch_maes.append(mae)
    print(np.mean(batch_maes))

evaluate_naive_method()
```

```
0.2897359729905486
```

Converting the MAE back to a Celsius error

In [9]:
```
celsius_mae = 0.29 * std[1]
celsius_mae
```

Out[9]:
```
2.5672247338393395
```

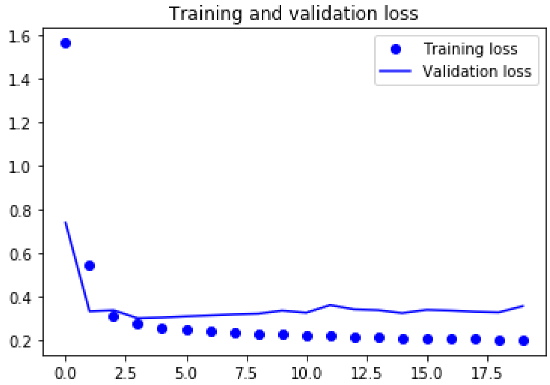
Training and evaluating a densely connected model

In [10]:
```
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.optimizers import RMSprop

model = Sequential()
model.add(Flatten(input_shape=(lookback // step, float_data.shape[-1])))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
model.summary()

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

```
Using TensorFlow backend.

WARNING:tensorflow:From C:\Users\pablo\AppData\Roaming\Python\Python36\site-packages\tensorflow\python\fra
mework\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will
be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_1 (Flatten)          (None, 3360)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 32)                107552    
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33        
=================================================================
Total params: 107,585
Trainable params: 107,585
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From C:\Users\pablo\AppData\Roaming\Python\Python36\site-packages\tensorflow\python\ops
\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a f
uture version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/20
500/500 [==============================] - 22s 44ms/step - loss: 1.5640 - val_loss: 0.7385
Epoch 2/20
500/500 [==============================] - 22s 44ms/step - loss: 0.5444 - val_loss: 0.3309
Epoch 3/20
500/500 [==============================] - 25s 51ms/step - loss: 0.3083 - val_loss: 0.3359
Epoch 4/20
500/500 [==============================] - 25s 50ms/step - loss: 0.2722 - val_loss: 0.2991
Epoch 5/20
500/500 [==============================] - 26s 51ms/step - loss: 0.2576 - val_loss: 0.3024
Epoch 6/20
500/500 [==============================] - 24s 48ms/step - loss: 0.2480 - val_loss: 0.3077s:
Epoch 7/20
500/500 [==============================] - 26s 53ms/step - loss: 0.2391 - val_loss: 0.3121
Epoch 8/20
500/500 [==============================] - 25s 50ms/step - loss: 0.2326 - val_loss: 0.3168
Epoch 9/20
500/500 [==============================] - 24s 47ms/step - loss: 0.2283 - val_loss: 0.3200
Epoch 10/20
500/500 [==============================] - 24s 49ms/step - loss: 0.2257 - val_loss: 0.3341
Epoch 11/20
500/500 [==============================] - 24s 47ms/step - loss: 0.2223 - val_loss: 0.3245
Epoch 12/20
500/500 [==============================] - 23s 47ms/step - loss: 0.2174 - val_loss: 0.3597
Epoch 13/20
500/500 [==============================] - 23s 46ms/step - loss: 0.2144 - val_loss: 0.3401
Epoch 14/20
500/500 [==============================] - 22s 44ms/step - loss: 0.2124 - val_loss: 0.3363
Epoch 15/20
500/500 [==============================] - 23s 46ms/step - loss: 0.2089 - val_loss: 0.3227
Epoch 16/20
500/500 [==============================] - 23s 46ms/step - loss: 0.2082 - val_loss: 0.3378
Epoch 17/20
500/500 [==============================] - 23s 46ms/step - loss: 0.2055 - val_loss: 0.3344
Epoch 18/20
500/500 [==============================] - 24s 47ms/step - loss: 0.2050 - val_loss: 0.3291
Epoch 19/20
500/500 [==============================] - 24s 47ms/step - loss: 0.2022 - val_loss: 0.3261
Epoch 20/20
500/500 [==============================] - 25s 50ms/step - loss: 0.1995 - val_loss: 0.3549
```

Plotting results

In [11]:
```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



A first recurrent baseline

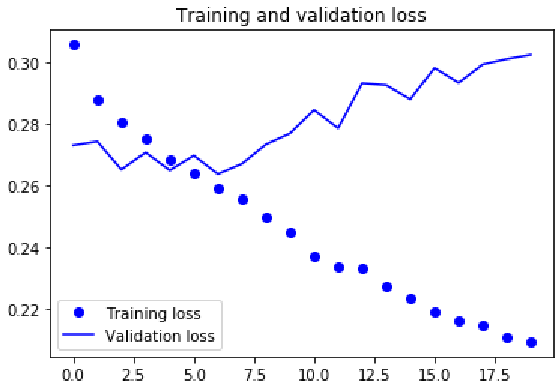
Training and evaluating a GRU-based model

In [12]:
```
from keras.models import Sequential
from keras.layers import GRU, Dense
from keras.optimizers import RMSprop

model = Sequential()
model.add(GRU(32, input_shape=(None, float_data.shape[-1])))
model.add(Dense(1))
model.summary()

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
gru_1 (GRU)                  (None, 32)                4512      
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33        
=================================================================
Total params: 4,545
Trainable params: 4,545
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
500/500 [==============================] - 99s 197ms/step - loss: 0.3057 - val_loss: 0.2730
Epoch 2/20
500/500 [==============================] - 96s 192ms/step - loss: 0.2878 - val_loss: 0.2743
Epoch 3/20
500/500 [==============================] - 96s 193ms/step - loss: 0.2804 - val_loss: 0.2651
Epoch 4/20
500/500 [==============================] - 94s 188ms/step - loss: 0.2752 - val_loss: 0.2707
Epoch 5/20
500/500 [==============================] - 92s 183ms/step - loss: 0.2683 - val_loss: 0.2649
Epoch 6/20
500/500 [==============================] - 94s 188ms/step - loss: 0.2638 - val_loss: 0.2697
Epoch 7/20
500/500 [==============================] - 94s 187ms/step - loss: 0.2593 - val_loss: 0.2637
Epoch 8/20
500/500 [==============================] - 94s 189ms/step - loss: 0.2555 - val_loss: 0.2670
Epoch 9/20
500/500 [==============================] - 92s 185ms/step - loss: 0.2500 - val_loss: 0.2733
Epoch 10/20
500/500 [==============================] - 94s 188ms/step - loss: 0.2448 - val_loss: 0.2769
Epoch 11/20
500/500 [==============================] - 94s 189ms/step - loss: 0.2371 - val_loss: 0.2845
Epoch 12/20
500/500 [==============================] - 96s 191ms/step - loss: 0.2337 - val_loss: 0.2785
Epoch 13/20
500/500 [==============================] - 92s 185ms/step - loss: 0.2331 - val_loss: 0.2931
Epoch 14/20
500/500 [==============================] - 94s 188ms/step - loss: 0.2277 - val_loss: 0.2925
Epoch 15/20
500/500 [==============================] - 94s 189ms/step - loss: 0.2236 - val_loss: 0.2879
Epoch 16/20
500/500 [==============================] - 94s 187ms/step - loss: 0.2194 - val_loss: 0.2981
Epoch 17/20
500/500 [==============================] - 93s 185ms/step - loss: 0.2165 - val_loss: 0.2933
Epoch 18/20
500/500 [==============================] - 93s 185ms/step - loss: 0.2146 - val_loss: 0.2992
Epoch 19/20
500/500 [==============================] - 95s 190ms/step - loss: 0.2107 - val_loss: 0.3010
Epoch 20/20
500/500 [==============================] - 63s 127ms/step - loss: 0.2095 - val_loss: 0.3024
```

Plotting results

In [13]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



Using recurrent dropout to fight overfitting

Training and evaluating a dropout-regularized GRU-based model

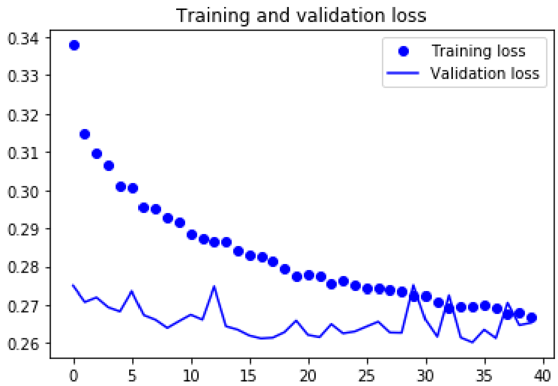
In [14]:
```
from keras.models import Sequential
from keras.layers import GRU, Dense
from keras.optimizers import RMSprop

model = Sequential()
model.add(GRU(32, dropout=0.2, recurrent_dropout=0.2, input_shape=(None, float_data.shape[-1])))
model.add(Dense(1))
model.summary()

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

```
WARNING:tensorflow:From C:\Users\pablo\Python\envs\DAVID\lib\site-packages\keras\backend\tensorflow_backen
d.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be re
moved in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
gru_2 (GRU)                  (None, 32)                4512      
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33        
=================================================================
Total params: 4,545
Trainable params: 4,545
Non-trainable params: 0
_________________________________________________________________
Epoch 1/40
500/500 [==============================] - 73s 145ms/step - loss: 0.3379 - val_loss: 0.2750
Epoch 2/40
500/500 [==============================] - 73s 146ms/step - loss: 0.3150 - val_loss: 0.2707
Epoch 3/40
500/500 [==============================] - 70s 140ms/step - loss: 0.3096 - val_loss: 0.2720
Epoch 4/40
500/500 [==============================] - 70s 140ms/step - loss: 0.3066 - val_loss: 0.2694
Epoch 5/40
500/500 [==============================] - 70s 140ms/step - loss: 0.3009 - val_loss: 0.2683
Epoch 6/40
500/500 [==============================] - 68s 137ms/step - loss: 0.3008 - val_loss: 0.2735
Epoch 7/40
500/500 [==============================] - 71s 141ms/step - loss: 0.2957 - val_loss: 0.2673
Epoch 8/40
500/500 [==============================] - 68s 136ms/step - loss: 0.2952 - val_loss: 0.2661
Epoch 9/40
500/500 [==============================] - 71s 142ms/step - loss: 0.2930 - val_loss: 0.2640
Epoch 10/40
500/500 [==============================] - 73s 147ms/step - loss: 0.2917 - val_loss: 0.2657
Epoch 11/40
500/500 [==============================] - 68s 137ms/step - loss: 0.2886 - val_loss: 0.2674
Epoch 12/40
500/500 [==============================] - 70s 139ms/step - loss: 0.2873 - val_loss: 0.2661
Epoch 13/40
500/500 [==============================] - 69s 138ms/step - loss: 0.2867 - val_loss: 0.2748
Epoch 14/40
500/500 [==============================] - 68s 137ms/step - loss: 0.2864 - val_loss: 0.2644
Epoch 15/40
500/500 [==============================] - 68s 137ms/step - loss: 0.2843 - val_loss: 0.2635
Epoch 16/40
500/500 [==============================] - 68s 135ms/step - loss: 0.2831 - val_loss: 0.2620
Epoch 17/40
500/500 [==============================] - 67s 135ms/step - loss: 0.2827 - val_loss: 0.2612
Epoch 18/40
500/500 [==============================] - 65s 130ms/step - loss: 0.2814 - val_loss: 0.2614
Epoch 19/40
500/500 [==============================] - 63s 127ms/step - loss: 0.2794 - val_loss: 0.2629
Epoch 20/40
500/500 [==============================] - 65s 130ms/step - loss: 0.2776 - val_loss: 0.2659
Epoch 21/40
500/500 [==============================] - 64s 128ms/step - loss: 0.2778 - val_loss: 0.2622
Epoch 22/40
500/500 [==============================] - 63s 125ms/step - loss: 0.2774 - val_loss: 0.2616
Epoch 23/40
500/500 [==============================] - 66s 131ms/step - loss: 0.2754 - val_loss: 0.2650
Epoch 24/40
500/500 [==============================] - 62s 124ms/step - loss: 0.2765 - val_loss: 0.2626
Epoch 25/40
500/500 [==============================] - 62s 124ms/step - loss: 0.2750 - val_loss: 0.2630
Epoch 26/40
500/500 [==============================] - 65s 130ms/step - loss: 0.2743 - val_loss: 0.2643
Epoch 27/40
500/500 [==============================] - 62s 123ms/step - loss: 0.2742 - val_loss: 0.2656
Epoch 28/40
500/500 [==============================] - 62s 125ms/step - loss: 0.2739 - val_loss: 0.2628
Epoch 29/40
500/500 [==============================] - 64s 128ms/step - loss: 0.2736 - val_loss: 0.2627
Epoch 30/40
500/500 [==============================] - 62s 124ms/step - loss: 0.2723 - val_loss: 0.2751
Epoch 31/40
500/500 [==============================] - 63s 127ms/step - loss: 0.2722 - val_loss: 0.2662
Epoch 32/40
500/500 [==============================] - 64s 129ms/step - loss: 0.2708 - val_loss: 0.2617
Epoch 33/40
```

```
500/500 [==============================] - 62s 124ms/step - loss: 0.2690 - val_loss: 0.2725
Epoch 34/40
500/500 [==============================] - 64s 127ms/step - loss: 0.2697 - val_loss: 0.2615
Epoch 35/40
500/500 [==============================] - 63s 127ms/step - loss: 0.2695 - val_loss: 0.2602
Epoch 36/40
500/500 [==============================] - 62s 124ms/step - loss: 0.2700 - val_loss: 0.2635
Epoch 37/40
500/500 [==============================] - 64s 128ms/step - loss: 0.2693 - val_loss: 0.2613
Epoch 38/40
500/500 [==============================] - 63s 125ms/step - loss: 0.2675 - val_loss: 0.2706
Epoch 39/40
500/500 [==============================] - 61s 123ms/step - loss: 0.2679 - val_loss: 0.2647
Epoch 40/40
500/500 [==============================] - 64s 128ms/step - loss: 0.2670 - val_loss: 0.2653
```

Plotting results

In [15]:
```python
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



Training and evaluating a dropout-regularized, stacked GRU model

In [16]:

```
model = Sequential()
model.add(GRU(32, dropout=0.1, recurrent_dropout=0.5, return_sequences=True, input_shape=(None, float_data
.shape[-1])))
model.add(GRU(64, activation='relu', dropout=0.1, recurrent_dropout=0.5))
model.add(Dense(1))
model.summary()

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

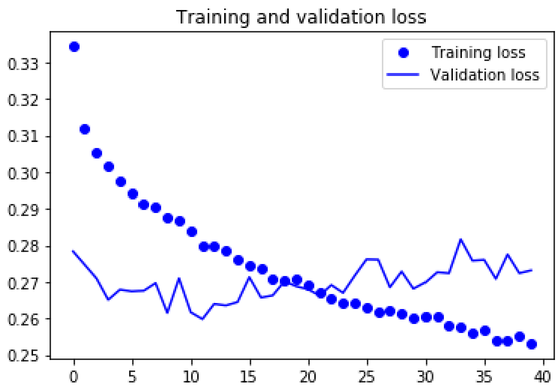
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_3 (GRU)                  (None, None, 32)          4512
_________________________________________________________________
gru_4 (GRU)                  (None, 64)                18624
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 23,201
Trainable params: 23,201
Non-trainable params: 0
_________________________________________________________________
Epoch 1/40
500/500 [==============================] - 225s 449ms/step - loss: 0.3343 - val_loss: 0.2783
Epoch 2/40
500/500 [==============================] - 223s 447ms/step - loss: 0.3116 - val_loss: 0.2746
Epoch 3/40
500/500 [==============================] - 224s 448ms/step - loss: 0.3053 - val_loss: 0.2709
Epoch 4/40
500/500 [==============================] - 224s 448ms/step - loss: 0.3015 - val_loss: 0.2651
Epoch 5/40
500/500 [==============================] - 224s 448ms/step - loss: 0.2974 - val_loss: 0.2679
Epoch 6/40
500/500 [==============================] - 224s 447ms/step - loss: 0.2940 - val_loss: 0.2674
Epoch 7/40
500/500 [==============================] - 223s 446ms/step - loss: 0.2913 - val_loss: 0.2676
Epoch 8/40
500/500 [==============================] - 223s 447ms/step - loss: 0.2903 - val_loss: 0.2697
Epoch 9/40
500/500 [==============================] - 222s 444ms/step - loss: 0.2878 - val_loss: 0.2615
Epoch 10/40
500/500 [==============================] - 223s 446ms/step - loss: 0.2869 - val_loss: 0.2710
Epoch 11/40
500/500 [==============================] - 223s 446ms/step - loss: 0.2841 - val_loss: 0.2617
Epoch 12/40
500/500 [==============================] - 233s 465ms/step - loss: 0.2800 - val_loss: 0.2598
Epoch 13/40
500/500 [==============================] - 238s 476ms/step - loss: 0.2799 - val_loss: 0.2639
Epoch 14/40
500/500 [==============================] - 230s 459ms/step - loss: 0.2786 - val_loss: 0.2635
Epoch 15/40
500/500 [==============================] - 230s 460ms/step - loss: 0.2762 - val_loss: 0.2645
Epoch 16/40
500/500 [==============================] - 229s 458ms/step - loss: 0.2746 - val_loss: 0.2713
Epoch 17/40
500/500 [==============================] - 233s 466ms/step - loss: 0.2735 - val_loss: 0.2657
Epoch 18/40
500/500 [==============================] - 233s 467ms/step - loss: 0.2709 - val_loss: 0.2663
Epoch 19/40
500/500 [==============================] - 233s 466ms/step - loss: 0.2705 - val_loss: 0.2701
Epoch 20/40
500/500 [==============================] - 232s 463ms/step - loss: 0.2706 - val_loss: 0.2688
Epoch 21/40
500/500 [==============================] - 228s 456ms/step - loss: 0.2691 - val_loss: 0.2680
Epoch 22/40
500/500 [==============================] - 229s 458ms/step - loss: 0.2670 - val_loss: 0.2662
Epoch 23/40
500/500 [==============================] - 230s 460ms/step - loss: 0.2655 - val_loss: 0.2691
Epoch 24/40
500/500 [==============================] - 228s 455ms/step - loss: 0.2644 - val_loss: 0.2670
Epoch 25/40
500/500 [==============================] - 230s 460ms/step - loss: 0.2640 - val_loss: 0.2717
Epoch 26/40
500/500 [==============================] - 230s 461ms/step - loss: 0.2629 - val_loss: 0.2762
Epoch 27/40
500/500 [==============================] - 236s 472ms/step - loss: 0.2615 - val_loss: 0.2761
Epoch 28/40
500/500 [==============================] - 250s 500ms/step - loss: 0.2622 - val_loss: 0.2685
Epoch 29/40
500/500 [==============================] - 251s 503ms/step - loss: 0.2612 - val_loss: 0.2728
Epoch 30/40
500/500 [==============================] - 230s 459ms/step - loss: 0.2600 - val_loss: 0.2681
Epoch 31/40
500/500 [==============================] - 229s 458ms/step - loss: 0.2604 - val_loss: 0.2698
Epoch 32/40
500/500 [==============================] - 229s 459ms/step - loss: 0.2604 - val_loss: 0.2726
Epoch 33/40
500/500 [==============================] - 236s 473ms/step - loss: 0.2581 - val_loss: 0.2724
Epoch 34/40
500/500 [==============================] - 230s 461ms/step - loss: 0.2578 - val_loss: 0.2817
```

```
Epoch 35/40
500/500 [==============================] - 231s 462ms/step - loss: 0.2561 - val_loss: 0.2758
Epoch 36/40
500/500 [==============================] - 231s 462ms/step - loss: 0.2566 - val_loss: 0.2760
Epoch 37/40
500/500 [==============================] - 232s 464ms/step - loss: 0.2541 - val_loss: 0.2708
Epoch 38/40
500/500 [==============================] - 227s 454ms/step - loss: 0.2538 - val_loss: 0.2775
Epoch 39/40
500/500 [==============================] - 233s 465ms/step - loss: 0.2553 - val_loss: 0.2724
Epoch 40/40
500/500 [==============================] - 234s 469ms/step - loss: 0.2531 - val_loss: 0.2731
```

Plotting results

In [17]:

```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



Using bidirectional RNNs

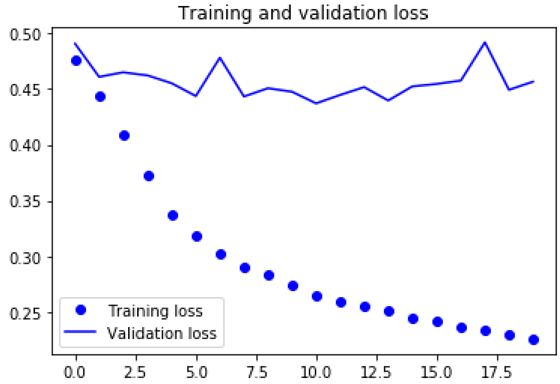
You need to do is write a variant of the data generator where the input sequences are reverted along the time dimension (replace the last line with yield samples[:, ::-1, :], targets).

In [18]:
```
def generator(data, lookback, delay, min_index, max_index,
              shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)

        samples = np.zeros((len(rows),
                           lookback // step,
                           data.shape[-1]))
        targets = np.zeros((len(rows),))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices]
            targets[j] = data[rows[j] + delay][1]
        yield samples[:, ::-1, :], targets

lookback = 1440
step = 6
delay = 144
batch_size = 128

train_gen = generator(float_data,
                      lookback=lookback,
                      delay=delay,
                      min_index=0,
                      max_index=200000,
                      shuffle=True,
                      step=step,
                      batch_size=batch_size)
val_gen = generator(float_data,
                    lookback=lookback,
                    delay=delay,
                    min_index=200001,
                    max_index=300000,
                    step=step,
                    batch_size=batch_size)
test_gen = generator(float_data,
                     lookback=lookback,
                     delay=delay,
                     min_index=300001,
                     max_index=None,
                     step=step,
                     batch_size=batch_size)

# This is how many steps to draw from `val_gen`
# in order to see the whole validation set:
val_steps = (300000 - 200001 - lookback) // batch_size

# This is how many steps to draw from `test_gen`
# in order to see the whole test set:
test_steps = (len(float_data) - 300001 - lookback) // batch_size
```

In [19]:

```
from keras.models import Sequential
from keras.layers import GRU, Dense
from keras.optimizers import RMSprop

model = Sequential()
model.add(GRU(32, input_shape=(None, float_data.shape[-1])))
model.add(Dense(1))
model.summary()

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_5 (GRU)                  (None, 32)                4512
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 33
=================================================================
Total params: 4,545
Trainable params: 4,545
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
500/500 [==============================] - 64s 128ms/step - loss: 0.4767 - val_loss: 0.4909
Epoch 2/20
500/500 [==============================] - 72s 144ms/step - loss: 0.4442 - val_loss: 0.4610
Epoch 3/20
500/500 [==============================] - 62s 125ms/step - loss: 0.4088 - val_loss: 0.4651
Epoch 4/20
500/500 [==============================] - 66s 132ms/step - loss: 0.3719 - val_loss: 0.4623
Epoch 5/20
500/500 [==============================] - 64s 128ms/step - loss: 0.3378 - val_loss: 0.4552
Epoch 6/20
500/500 [==============================] - 60s 120ms/step - loss: 0.3184 - val_loss: 0.4437
Epoch 7/20
500/500 [==============================] - 62s 125ms/step - loss: 0.3021 - val_loss: 0.4783
Epoch 8/20
500/500 [==============================] - 61s 123ms/step - loss: 0.2908 - val_loss: 0.4433
Epoch 9/20
500/500 [==============================] - 60s 121ms/step - loss: 0.2828 - val_loss: 0.4508
Epoch 10/20
500/500 [==============================] - 61s 123ms/step - loss: 0.2741 - val_loss: 0.4476
Epoch 11/20
500/500 [==============================] - 62s 124ms/step - loss: 0.2649 - val_loss: 0.4372
Epoch 12/20
500/500 [==============================] - 61s 121ms/step - loss: 0.2593 - val_loss: 0.4447
Epoch 13/20
500/500 [==============================] - 63s 127ms/step - loss: 0.2554 - val_loss: 0.4518
Epoch 14/20
500/500 [==============================] - 61s 123ms/step - loss: 0.2510 - val_loss: 0.4397
Epoch 15/20
500/500 [==============================] - 60s 120ms/step - loss: 0.2448 - val_loss: 0.4524
Epoch 16/20
500/500 [==============================] - 63s 125ms/step - loss: 0.2410 - val_loss: 0.4547
Epoch 17/20
500/500 [==============================] - 61s 122ms/step - loss: 0.2362 - val_loss: 0.4577
Epoch 18/20
500/500 [==============================] - 60s 119ms/step - loss: 0.2340 - val_loss: 0.4921
Epoch 19/20
500/500 [==============================] - 63s 126ms/step - loss: 0.2294 - val_loss: 0.4494
Epoch 20/20
500/500 [==============================] - 61s 123ms/step - loss: 0.2256 - val_loss: 0.4567
```



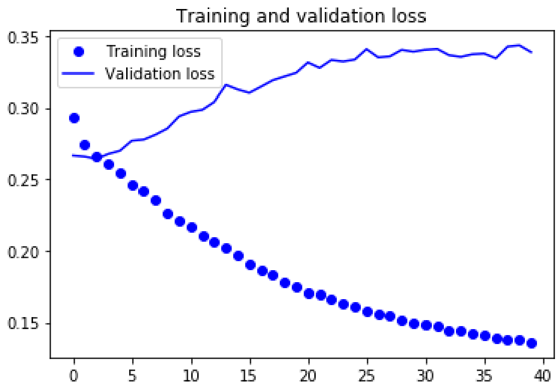
Training a bidirectional GRU

In [23]:

```
from keras.layers import Bidirectional

model = Sequential()
model.add(Bidirectional(GRU(32), input_shape=(None, float_data.shape[-1])))
model.add(Dense(1))
model.summary()

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_2 (Bidirection (None, 64)                9024
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 65
=================================================================
Total params: 9,089
Trainable params: 9,089
Non-trainable params: 0
_________________________________________________________________
Epoch 1/40
500/500 [==============================] - 92s 185ms/step - loss: 0.2929 - val_loss: 0.2665
Epoch 2/40
500/500 [==============================] - 95s 190ms/step - loss: 0.2746 - val_loss: 0.2658
Epoch 3/40
500/500 [==============================] - 93s 186ms/step - loss: 0.2661 - val_loss: 0.2642
Epoch 4/40
500/500 [==============================] - 90s 179ms/step - loss: 0.2603 - val_loss: 0.2677
Epoch 5/40
500/500 [==============================] - 91s 181ms/step - loss: 0.2543 - val_loss: 0.2699
Epoch 6/40
500/500 [==============================] - 88s 176ms/step - loss: 0.2465 - val_loss: 0.2769
Epoch 7/40
500/500 [==============================] - 91s 183ms/step - loss: 0.2423 - val_loss: 0.2777
Epoch 8/40
500/500 [==============================] - 89s 177ms/step - loss: 0.2351 - val_loss: 0.2812
Epoch 9/40
500/500 [==============================] - 90s 181ms/step - loss: 0.2265 - val_loss: 0.2854
Epoch 10/40
500/500 [==============================] - 88s 177ms/step - loss: 0.2209 - val_loss: 0.2939
Epoch 11/40
500/500 [==============================] - 90s 180ms/step - loss: 0.2163 - val_loss: 0.2971
Epoch 12/40
500/500 [==============================] - 88s 176ms/step - loss: 0.2104 - val_loss: 0.2985
Epoch 13/40
500/500 [==============================] - 90s 179ms/step - loss: 0.2064 - val_loss: 0.3039
Epoch 14/40
500/500 [==============================] - 87s 175ms/step - loss: 0.2023 - val_loss: 0.3158
Epoch 15/40
500/500 [==============================] - 90s 179ms/step - loss: 0.1971 - val_loss: 0.3125
Epoch 16/40
500/500 [==============================] - 87s 174ms/step - loss: 0.1907 - val_loss: 0.3102
Epoch 17/40
500/500 [==============================] - 90s 179ms/step - loss: 0.1865 - val_loss: 0.3145
Epoch 18/40
500/500 [==============================] - 87s 175ms/step - loss: 0.1831 - val_loss: 0.3189
Epoch 19/40
500/500 [==============================] - 90s 180ms/step - loss: 0.1781 - val_loss: 0.3216
Epoch 20/40
500/500 [==============================] - 88s 175ms/step - loss: 0.1748 - val_loss: 0.3241
Epoch 21/40
500/500 [==============================] - 90s 180ms/step - loss: 0.1707 - val_loss: 0.3314
Epoch 22/40
500/500 [==============================] - 88s 176ms/step - loss: 0.1691 - val_loss: 0.3276
Epoch 23/40
500/500 [==============================] - 90s 180ms/step - loss: 0.1666 - val_loss: 0.3331
Epoch 24/40
500/500 [==============================] - 88s 175ms/step - loss: 0.1628 - val_loss: 0.3321
Epoch 25/40
500/500 [==============================] - 90s 180ms/step - loss: 0.1610 - val_loss: 0.3334
Epoch 26/40
500/500 [==============================] - 86s 172ms/step - loss: 0.1580 - val_loss: 0.3407
Epoch 27/40
500/500 [==============================] - 89s 177ms/step - loss: 0.1561 - val_loss: 0.3349
Epoch 28/40
500/500 [==============================] - 86s 172ms/step - loss: 0.1544 - val_loss: 0.3356
Epoch 29/40
500/500 [==============================] - 88s 177ms/step - loss: 0.1521 - val_loss: 0.3402
Epoch 30/40
500/500 [==============================] - 86s 172ms/step - loss: 0.1496 - val_loss: 0.3389
Epoch 31/40
500/500 [==============================] - 88s 176ms/step - loss: 0.1485 - val_loss: 0.3402
Epoch 32/40
500/500 [==============================] - 86s 172ms/step - loss: 0.1477 - val_loss: 0.3408
Epoch 33/40
500/500 [==============================] - 87s 175ms/step - loss: 0.1447 - val_loss: 0.3365
Epoch 34/40
500/500 [==============================] - 86s 172ms/step - loss: 0.1439 - val_loss: 0.3353
Epoch 35/40
500/500 [==============================] - 87s 174ms/step - loss: 0.1425 - val_loss: 0.3371
```

```
Epoch 36/40
500/500 [==============================] - 87s 173ms/step - loss: 0.1409 - val_loss: 0.3376
Epoch 37/40
500/500 [==============================] - 87s 174ms/step - loss: 0.1392 - val_loss: 0.3343
Epoch 38/40
500/500 [==============================] - 87s 174ms/step - loss: 0.1379 - val_loss: 0.3425
Epoch 39/40
500/500 [==============================] - 87s 174ms/step - loss: 0.1376 - val_loss: 0.3434
Epoch 40/40
500/500 [==============================] - 87s 174ms/step - loss: 0.1360 - val_loss: 0.3386
```



Pablo Minango

- pablodavid218@gmail.com