File Edit View Insert Cell Kernel Navigate Widgets Help

Not Trusted | Python 3 (ipykernel)

Run | Markdown

錯誤來源:

Listing 6.37. Training and evaluating a densely connected model - stalls on first epoch #49

Closed jswift24 opened this issue on Apr 19, 2018 · 2 comments

jswift24 commented on Apr 19, 2018 • edited

Having trouble getting listing 6.37 to work. The model stalls on the first epoch. Getting the following output, but it never reaches the end of epoch 1:

```
Epoch 1/20
496/500 [============================>.] - ETA: 0s - loss: 1.2985
```

Prior code in this chapter and code in prior chapters works fine. Any suggestions?

Thanks,

driesbuyck commented on May 1, 2018

In the book, listing 6.34 has an error. The last 2 steps need // batch_size.

參考1github
參考2kaggle

小小抱怨一下: 這次課本的錯誤太多了拉，搞得我懷疑人生。

- 雖然找尋錯誤和找答案也是蠻重要的一環就是了。

In [1]:
```
import os
data_dir = 'C:/Users/Huang/Downloads/jena_climate'
fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')
```

In [2]:
```
f = open(fname)
data = f.read()
f.close()
lines = data.split('\n')
header = lines[0].split(',')
lines = lines[1:]
print(header)
print(len(lines))
#convert all 420,551 lines of data into a Numpy array
```

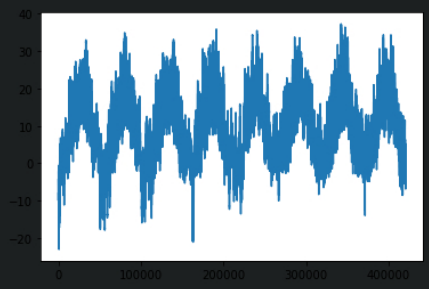
```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"rh (%)"', '"VPmax (mbar)"', '"VPact (mbar)"', '"VPdef (mbar)"', '"sh (g/kg)"', '"H2OC (mmol/mol)"', '"rho (g/m**3)"', '"wv (m/s)"', '"max. wv (m/s)"', '"wd (deg)"']
420451
```

1 Listing 6.29. Parsing the data

In [3]:
```
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
```

2 Listing 6.30. Plotting the temperature timeseries

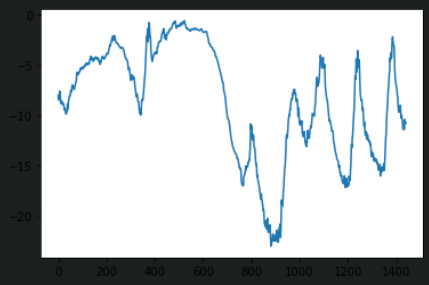
In [4]:
```
from matplotlib import pyplot as plt

# Temperature over the full temporal range of the dataset. Yearly trends are clearly visible

temp = float_data[:, 1] # <1> temperature (in degrees Celsius)
plt.plot(range(len(temp)), temp)
plt.show()
```



3 Listing 6.31. Plotting the first 10 days of the temperature timese

In [5]:
```
plt.plot(range(1440), temp[:1440])
```

```
[<matplotlib.lines.Line2D at 0x1483851e850>]
```



4 Listing 6.32. Normalizing the data

In [6]:
```
mean = float_data[:200000].mean(axis=0)
float_data -= mean
std = float_data[:200000].std(axis=0)
float_data /= std
float_data
```

```
array([[ 0.90014748, -1.93135845, -1.98211036, ..., -0.72950452,
        -0.78067973, -0.27613603],
       [ 0.9060434 , -1.97541381, -2.02567   , ..., -0.93124017,
        -0.88794488, -0.46317443],
       [ 0.90132666, -1.98671006, -2.03683914, ..., -1.27614304,
        -1.26122763, -0.05330633],
       ...,
       [ 1.28927851, -1.38236094, -1.46721307, ..., -0.69696652,
        -0.67341457,  0.45008097],
       [ 1.28809932, -1.50323076, -1.58672285, ..., -0.43015486,
        -0.60476487,  0.57246412],
       [ 1.28927851, -1.56987861, -1.65150386, ..., -0.5993525 ,
        -0.690577  ,  0.10024989]])
```

5 Listing 6.33. Generator yielding timeseries samples and their targets

In [7]:
```
# generator function used to feed the training, validation and test data

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
```

6 Listing 6.34. Preparing the training, validation, and test generators

- 這裡開始作者就開始怪怪的/都少打

In [8]:

```
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 # How many steps to draw from
          # val_gen in order to see the entire validation set
test_steps = (len(float_data) - 300001 - lookback)  // batch_size # How many steps to draw
        # from test_gen in order to see the entire test set
```

In [9]:

```
val_steps
```

```
769
```

In [10]:

```
test_steps
```

```
929
```

7 Listing 6.35. Computing the common-sense baseline MAE

In [11]:

```
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))
```

8 Listing 6.36. Converting the MAE back to a Celsius error

In [35]:
```
celsius_mae = 0.29 * std[1]
celsius_mae
```

```
2.5672247338393395
```

9 Listing 6.37. Training and evaluating a densely connected model

In [36]:
```
float_data.shape[-1]
```

```
14
```

In [37]:
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

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

```
Epoch 1/20

C:\Users\Huang\AppData\Local\Temp\ipykernel_3940\2841118526.py:12: UserWarning: `Model.fit_generator` is deprecated and will be
removed in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(train_gen,

500/500 [==============================] - 17s 31ms/step - loss: 1.1192 - val_loss: 0.6695
Epoch 2/20
500/500 [==============================] - 15s 30ms/step - loss: 0.3826 - val_loss: 0.3132
Epoch 3/20
500/500 [==============================] - 15s 30ms/step - loss: 0.2847 - val_loss: 0.3012
Epoch 4/20
500/500 [==============================] - 19s 38ms/step - loss: 0.2679 - val_loss: 0.3144
Epoch 5/20
500/500 [==============================] - 17s 35ms/step - loss: 0.2536 - val_loss: 0.3152
Epoch 6/20
500/500 [==============================] - 15s 30ms/step - loss: 0.2472 - val_loss: 0.3115
Epoch 7/20
500/500 [==============================] - 15s 30ms/step - loss: 0.2396 - val_loss: 0.3078
Epoch 8/20
500/500 [==============================] - 15s 30ms/step - loss: 0.2341 - val_loss: 0.3259
```

10 Listing 6.38. Plotting results

In [38]:
```
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
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()
```

可以發現效果很差

11 Listing 6.39. Training and evaluating a GRU-based model

In [40]:

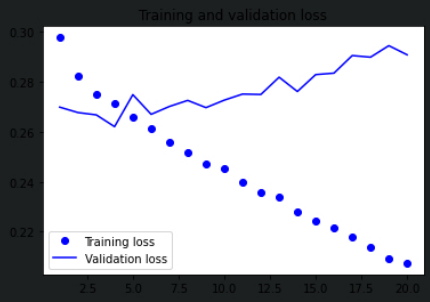
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

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

```
Epoch 1/20

C:\Users\Huang\AppData\Local\Temp\ipykernel_3940\1554876590.py:9: UserWarning: `Model.fit_generator` is deprecated and will be rem
oved in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(train_gen,

500/500 [==============================] - 99s 190ms/step - loss: 0.2978 - val_loss: 0.2699
Epoch 2/20
500/500 [==============================] - 93s 186ms/step - loss: 0.2824 - val_loss: 0.2678
Epoch 3/20
500/500 [==============================] - 101s 203ms/step - loss: 0.2751 - val_loss: 0.2668
Epoch 4/20
500/500 [==============================] - 98s 196ms/step - loss: 0.2716 - val_loss: 0.2621
Epoch 5/20
500/500 [==============================] - 95s 190ms/step - loss: 0.2660 - val_loss: 0.2749
Epoch 6/20
500/500 [==============================] - 92s 184ms/step - loss: 0.2612 - val_loss: 0.2670
Epoch 7/20
500/500 [==============================] - 91s 182ms/step - loss: 0.2561 - val_loss: 0.2702
Epoch 8/20
500/500 [==============================] - 93s 186ms/step - loss: 0.2517 - val_loss: 0.2726
Epoch 9/20
500/500 [==============================] - 93s 185ms/step - loss: 0.2470 - val_loss: 0.2697
Epoch 10/20
500/500 [==============================] - 97s 193ms/step - loss: 0.2455 - val_loss: 0.2727
Epoch 11/20
500/500 [==============================] - 93s 187ms/step - loss: 0.2397 - val_loss: 0.2752
Epoch 12/20
500/500 [==============================] - 95s 190ms/step - loss: 0.2359 - val_loss: 0.2750
Epoch 13/20
500/500 [==============================] - 93s 187ms/step - loss: 0.2339 - val_loss: 0.2819
Epoch 14/20
500/500 [==============================] - 94s 188ms/step - loss: 0.2279 - val_loss: 0.2762
Epoch 15/20
500/500 [==============================] - 93s 186ms/step - loss: 0.2243 - val_loss: 0.2830
Epoch 16/20
500/500 [==============================] - 95s 191ms/step - loss: 0.2217 - val_loss: 0.2835
Epoch 17/20
500/500 [==============================] - 96s 191ms/step - loss: 0.2180 - val_loss: 0.2906
Epoch 18/20
500/500 [==============================] - 94s 189ms/step - loss: 0.2138 - val_loss: 0.2899
Epoch 19/20
500/500 [==============================] - 93s 186ms/step - loss: 0.2091 - val_loss: 0.2946
Epoch 20/20
500/500 [==============================] - 95s 191ms/step - loss: 0.2074 - val_loss: 0.2909
```

In [41]:

```
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
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()
```



在4位置也嚴重地overfitting

12 Listing 6.40. Training and evaluating a dropout-regularized GRU-based model

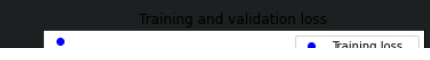
In [42]:

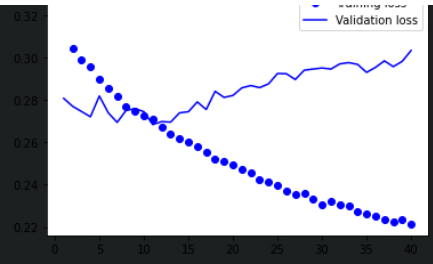
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop
model = Sequential()
model.add(layers.GRU(32,
    dropout=0.2,
    recurrent_dropout=0.2,
    input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))
model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
    steps_per_epoch=500,
    epochs=40,
    validation_data=val_gen,
    validation_steps=val_steps)
```

```
Epoch 1/40

C:\Users\Huang\AppData\Local\Temp\ipykernel_3940\768164218.py:11: UserWarning: `Model.fit_generator` is deprecated and will be r
emoved in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(train_gen,


500/500 [==============================] - 155s 295ms/step - loss: 0.3282 - val_loss: 0.2809
Epoch 2/40
500/500 [==============================] - 149s 299ms/step - loss: 0.3048 - val_loss: 0.2772
Epoch 3/40
500/500 [==============================] - 146s 292ms/step - loss: 0.2994 - val_loss: 0.2747
Epoch 4/40
500/500 [==============================] - 146s 293ms/step - loss: 0.2959 - val_loss: 0.2722
Epoch 5/40
500/500 [==============================] - 149s 297ms/step - loss: 0.2900 - val_loss: 0.2820
Epoch 6/40
500/500 [==============================] - 147s 294ms/step - loss: 0.2857 - val_loss: 0.2741
Epoch 7/40
500/500 [==============================] - 148s 296ms/step - loss: 0.2821 - val_loss: 0.2695
Epoch 8/40
500/500 [==============================] - 148s 295ms/step - loss: 0.2769 - val_loss: 0.2752
```

In [43]:

```
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
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()
```





13 Listing 6.41. Training and evaluating a dropout-regularized, stacked GRU model

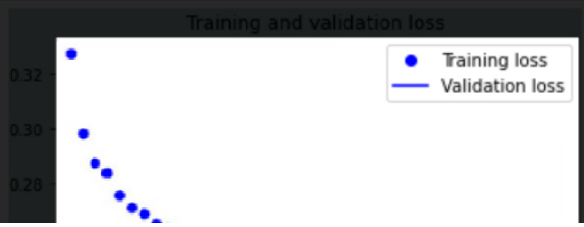
In [ ]:

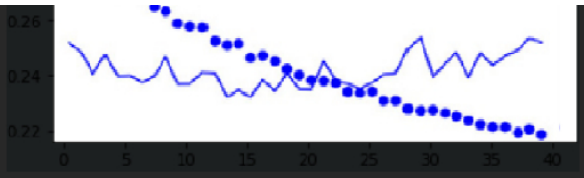
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop
model = Sequential()
model.add(layers.GRU(32,
    dropout=0.1,
    recurrent_dropout=0.5,
    return_sequences=True,
    input_shape=(None, float_data.shape[-1])))
model.add(layers.GRU(64, activation='relu',
    dropout=0.1,
    recurrent_dropout=0.5))
model.add(layers.Dense(1))
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)
```

```
Epoch 1/40


C:\Users\Huang\AppData\Local\Temp\ipykernel_3940\952095746.py:15: UserWarning: `Model.fit_generator` is deprecated and will be r
emoved in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(train_gen,


500/500 [==============================] - 483s 938ms/step - loss: 0.3206 - val_loss: 0.2791
Epoch 2/40
500/500 [==============================] - 454s 908ms/step - loss: 0.3027 - val_loss: 0.2725
Epoch 3/40
500/500 [==============================] - 447s 894ms/step - loss: 0.2973 - val_loss: 0.2710
Epoch 4/40
500/500 [==============================] - 440s 881ms/step - loss: 0.2899 - val_loss: 0.2762
Epoch 5/40
500/500 [==============================] - 439s 878ms/step - loss: 0.2842 - val_loss: 0.2746
Epoch 6/40
500/500 [==============================] - 444s 888ms/step - loss: 0.2794 - val_loss: 0.2662
Epoch 7/40
500/500 [==============================] - 452s 904ms/step - loss: 0.2730 - val_loss: 0.2762
Epoch 8/40
500/500 [==============================] - 460s 921ms/step - loss: 0.2709 - val_loss: 0.2729
```

In [ ]:

```
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
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()
```





overfitting有改善

14 Listing 6.42. Training and evaluating an LSTM using reversed sequences

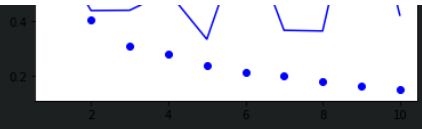
In [14]:

```
from keras.datasets import imdb
#from keras.preprocessing import sequence
from keras.utils.data_utils import pad_sequences
from keras import layers
from keras.models import Sequential
max_features = 10000
maxlen = 500
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = [x[::-1] for x in x_train]
x_test = [x[::-1] for x in x_test]
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
model = Sequential()
model.add(layers.Embedding(max_features, 128))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
    loss='binary_crossentropy',
    metrics=['acc'])
history = model.fit(x_train, y_train,
    epochs=10,
    batch_size=128,
    validation_split=0.2)
```

```
Epoch 1/10
157/157 [==============================] - 84s 517ms/step - loss: 0.6098 - acc: 0.6486 - val_loss: 0.6787 - val_acc: 0.6754
Epoch 2/10
157/157 [==============================] - 90s 573ms/step - loss: 0.4022 - acc: 0.8393 - val_loss: 0.4383 - val_acc: 0.8094
Epoch 3/10
157/157 [==============================] - 93s 593ms/step - loss: 0.3092 - acc: 0.8817 - val_loss: 0.4393 - val_acc: 0.7934
Epoch 4/10
157/157 [==============================] - 85s 541ms/step - loss: 0.2800 - acc: 0.8950 - val_loss: 0.5068 - val_acc: 0.7640
Epoch 5/10
157/157 [==============================] - 86s 548ms/step - loss: 0.2371 - acc: 0.9148 - val_loss: 0.3329 - val_acc: 0.8586
Epoch 6/10
157/157 [==============================] - 85s 538ms/step - loss: 0.2119 - acc: 0.9256 - val_loss: 0.7404 - val_acc: 0.7766
Epoch 7/10
157/157 [==============================] - 86s 547ms/step - loss: 0.2000 - acc: 0.9293 - val_loss: 0.3658 - val_acc: 0.8648
Epoch 8/10
157/157 [==============================] - 86s 548ms/step - loss: 0.1778 - acc: 0.9408 - val_loss: 0.3632 - val_acc: 0.8604
Epoch 9/10
157/157 [==============================] - 86s 551ms/step - loss: 0.1593 - acc: 0.9450 - val_loss: 0.9775 - val_acc: 0.7610
Epoch 10/10
157/157 [==============================] - 96s 610ms/step - loss: 0.1487 - acc: 0.9511 - val_loss: 0.4215 - val_acc: 0.8736
```

In [16]:

```
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
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()
```



相較上個，改善不少。速度也挺快的。

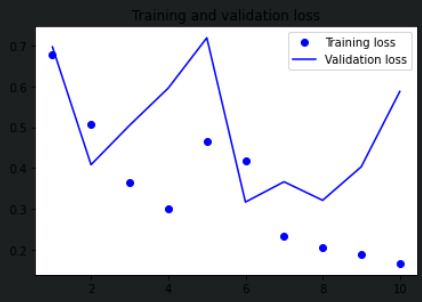
15 Listing 6.43. Training and evaluating a bidirectional LSTM

In [ ]:

```
model = Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.Bidirectional(layers.LSTM(32)))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train,
 epochs=10,
 batch_size=128,
 validation_split=0.2)
```

In [18]:

```
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
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()
```



仍然有點overfitting，只是速度更快。

Listing 6.44. Training a bidirectional GRU

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop
model = Sequential()
model.add(layers.Bidirectional(
 layers.GRU(32), input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))
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)
```

最後一個就不執行因為不適合(用未來去推測過去)