PythonとKerasによるディープラーニング

第6章　テキストとシーケンスのためのディープラーニング
6.4　畳み込みニューラルネットワークでのシーケンス処理

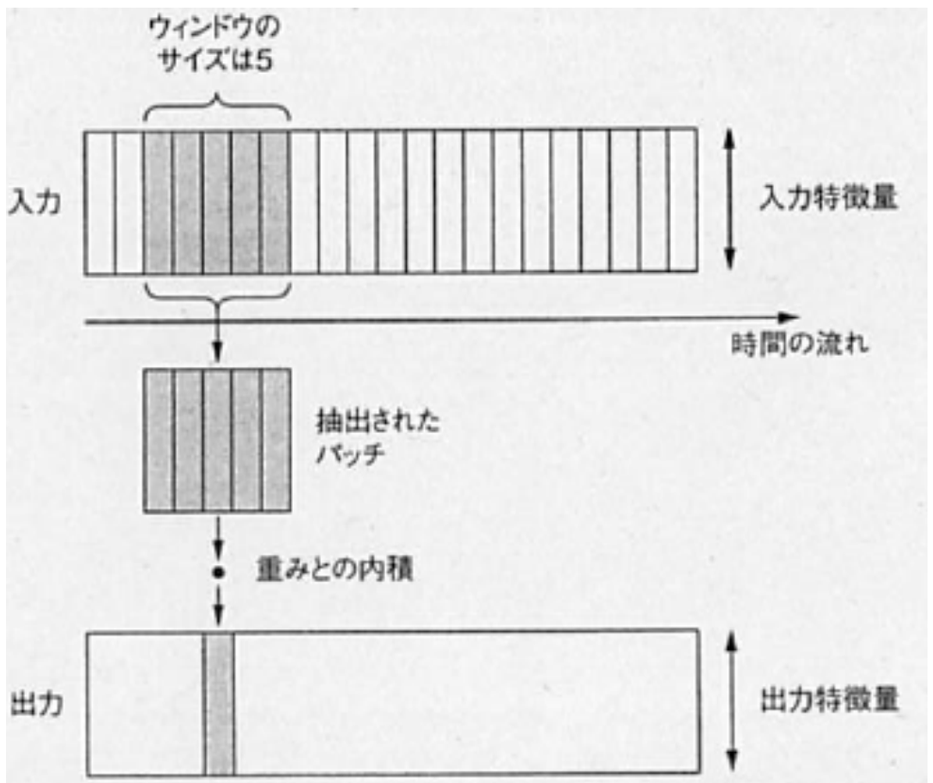
19nm715n ZHANGYIHANG

第５章では、畳み込みニューラルネットワークを取り上げ、CNNが特定にコンピュータビジョンの問題でうまくいくことを説明しました。今回は1次元CNNでシーケンス処理問題において説明します。

6.4.1　シーケンスデータでの1次元CNN

　1次元CNNは、シーケンスから局所的な１次元パッチを抽出できます。こうした１次元の畳み込み層では、シーケンスから局所的なパターンを認識できます。各パッチの入力変換が同じため、シーケンスの特定の位置から学習したパターンを後から別の位置で認識することが可能です。このため、1次元CNNが学習するパターンは移動不変となります。

- １次元CNNでは、入力シーケンスの時間的なパッチから出力として時間刻みが取得される。



6.4.2　シーケンスデータの1次元プーリング

　1次元のCNNは、２次元のプーリング演算に相当するプーリング演算であり、入力から1次元パッチを抽出し、最大値または平均値を出力します。２次元のCNNと同様に、この演算は１次元の入力の長さを削減するために使用されます。

6.4.3　1次元CNNの実装

Kerasで１次元CNNを実装するには、Conv1D層を使用します。

- 単純な2層の１次元CNNを構築し、IMDbの感情分析タスクに適用してみます。

リスト 6-45：IMDb データの準備

```
from keras.datasets import imdb
from keras.preprocessing import sequence

max_features = 10000  # 特徴量として考慮する単語の数
max_len = 500         # この数の単語を残してテキストをカット

print('Loading data...')
(x_train, y_train), (x_test, y_test) = \
    imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')

print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

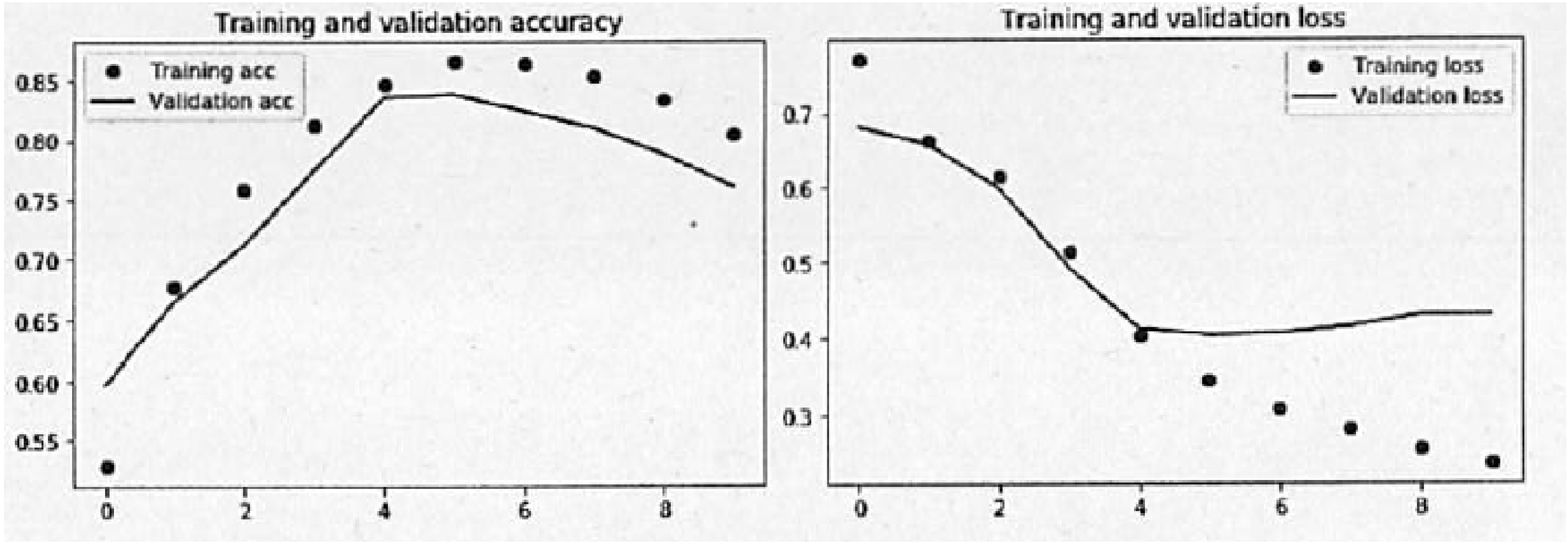
- **IMDbデータセットでの単純な１次元CNNの訓練と評価**

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

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
model.summary()

model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

- **訓練と検証の結果**



　単語レベルの感情分析タスクでは、１次元CNNがRNNに代わる高速で安価なアプローチになり得ることが実証されています。

6.4.4　CNNとRNNを組み合わせて長いシーケンスを処理する

　１次元CNNはRNNと異なり、時間刻みの順序に敏感ではありません。この弱さを明らかにするために、気温予測問題で１次元CNNを試してみます。

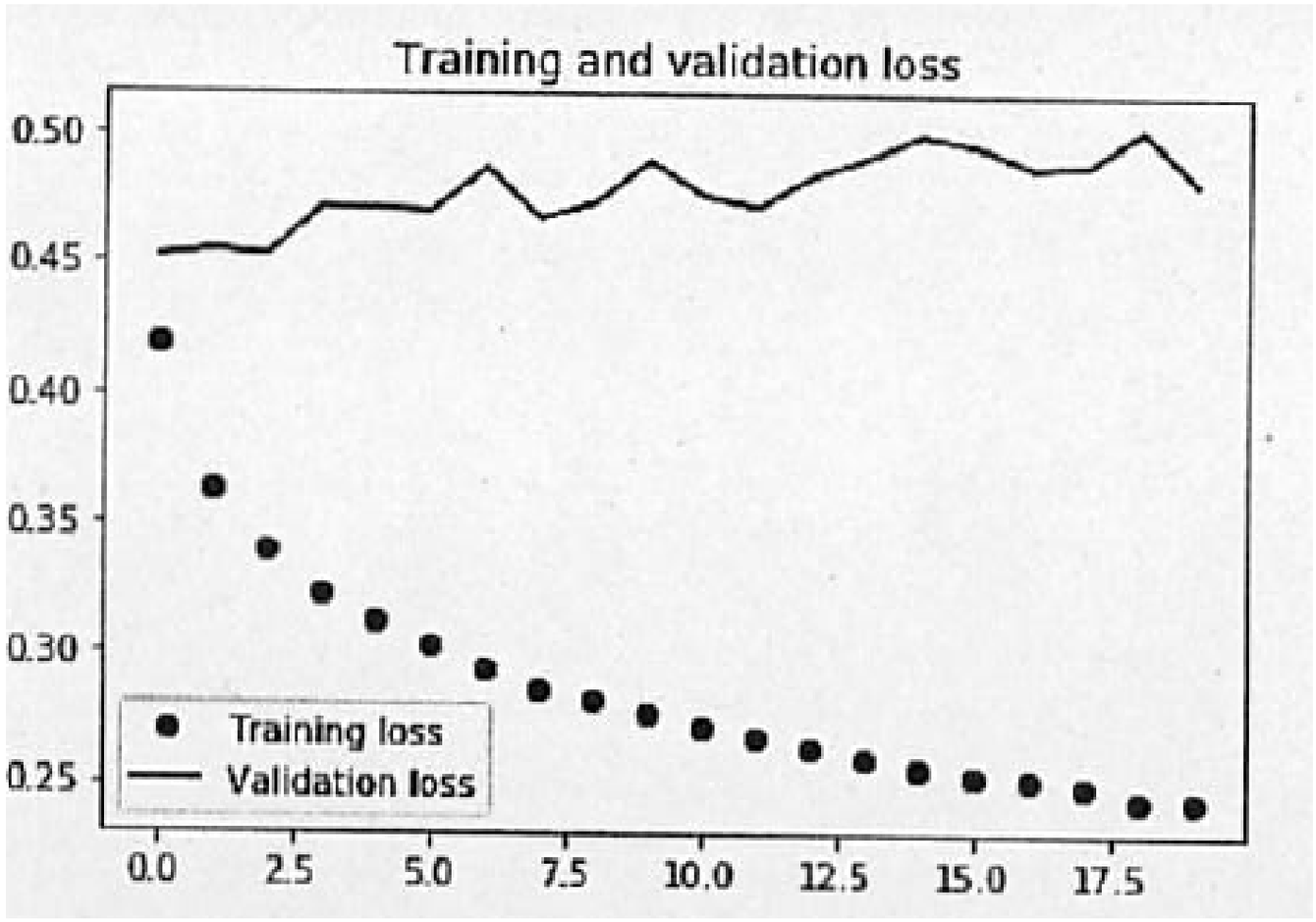
リスト 6-47：気象データセットでの単純な 1 次元 CNN の訓練と評価

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

model = Sequential()
model.add(layers.Conv1D(32, 5, activation='relu',
                        input_shape=(None, float_data.shape[-1])))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(32, 5, activation='relu'))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(32, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
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)
```

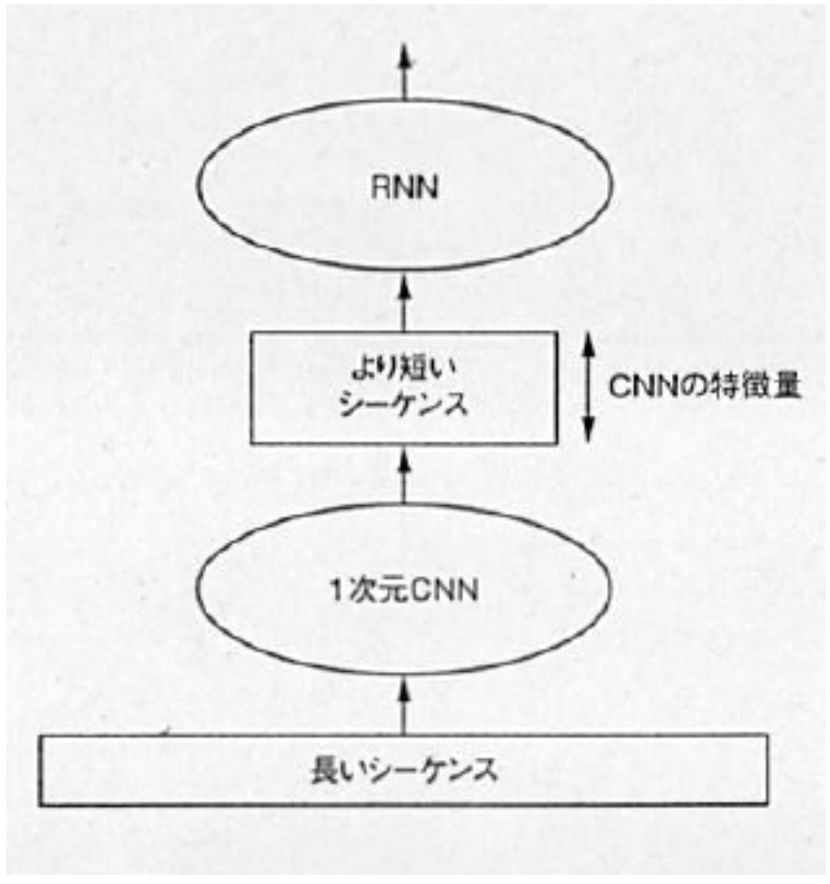
- **気温予測タスクで単純な１次元CNNを使用した場合の損失値**



　検証データでの平均絶対誤差は0.40台であり、この小さなCNNでは、常識的なベースラインすらクリアできません。

- **CNNとRNNの組み合わせ**

　CNNの軽快さをRNNの順序への敏感さと組み合わせる手法の1つは、１次元CNNをRNNの前処理ステップとして使用します。



　この手法は、特にRNNで処理出来ないほど長いシーケンスを実現している場合です。

- **Jenaデータセット用のより分解能の高いデータジェネレータの準備**

```
step = 3          # 前回は6 (1時間に1データ点)、今回は3 (30分に1データ点)
lookback = 720    # 変更なし
delay = 144       # 変更なし

train_gen = generator(float_data,
                      lookback=lookback,
                      delay=delay,
                      min_index=0,
                      max_index=200000,
                      shuffle=True,
                      step=step)
val_gen = generator(float_data,
                    lookback=lookback,
                    delay=delay,
                    min_index=200001,
                    max_index=300000,
                    step=step)
test_gen = generator(float_data,
                     lookback=lookback,
                     delay=delay,
                     min_index=300001,
                     max_index=None,
                     step=step)
val_steps = (300000 - 200001 - lookback) // 128
test_steps = (len(float_data) - 300001 - lookback) // 128
```

- **1次元畳み込みベースとGRU層で構成された**

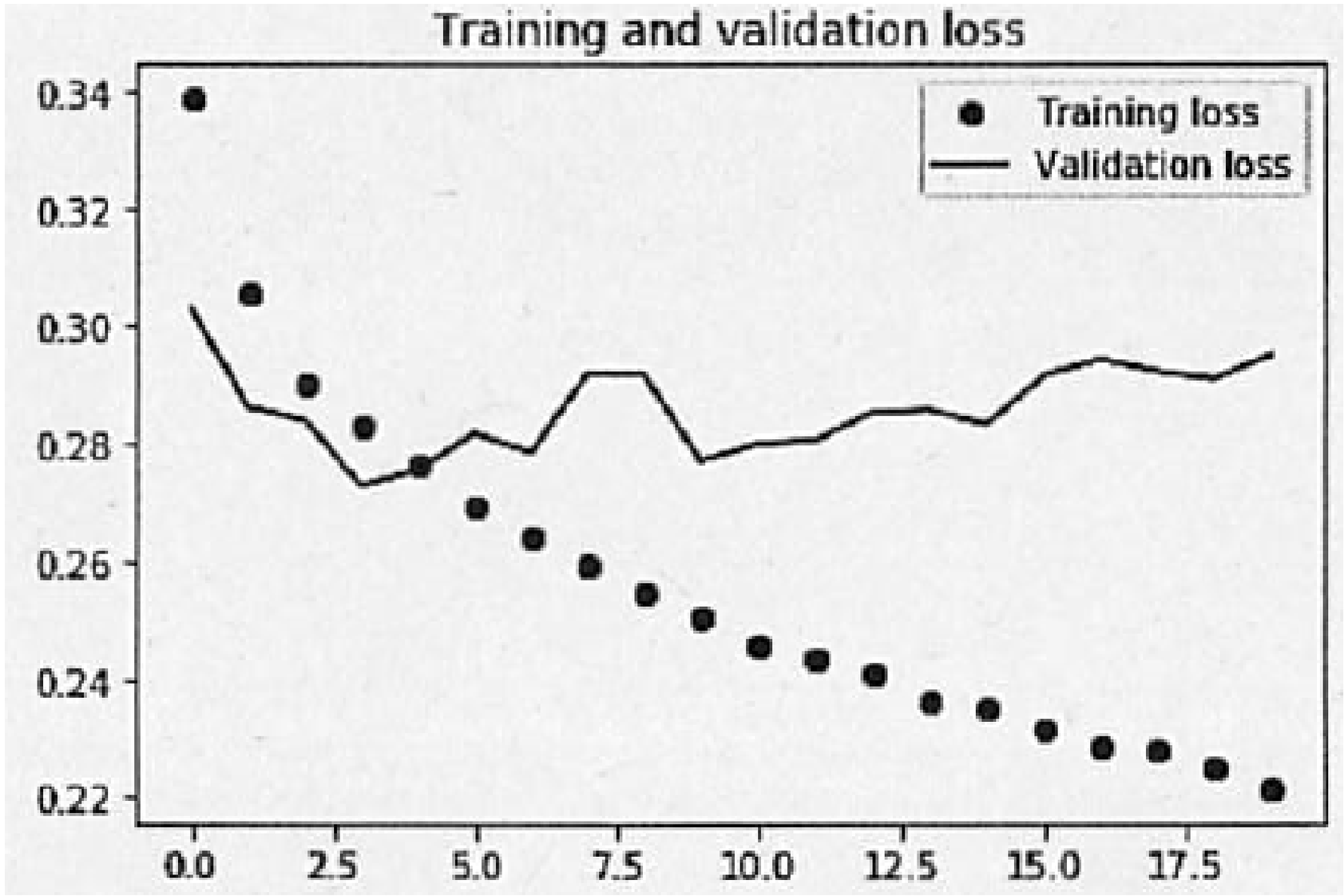
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Conv1D(32, 5, activation='relu',
                        input_shape=(None, float_data.shape[-1])))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(32, 5, activation='relu'))
model.add(layers.GRU(32, dropout=0.1, recurrent_dropout=0.5))
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)
```

- **気温予測問題で１次元CNNとGRUを使用した場合の損失値**



検証データでの損失値から判断すると、この設定は正則化したGRUベースのモデルほどよくありませんが、かなり改善されています。

6.4.5　まとめ

- ２次元CNNが２次元空間での視覚パターンの処理に適しているのと同様に、１次元CNNは時間的なパターンの処理に適している。
- 一般に、1次元CNNの構造は、コンピュータビジョン分野の２次元CNNとほぼ同じである。
- 非常に長いシーケンスをRNNで処理するとかなりコストがかかるが、１次元CNNではそれほどコストがかからない。このため、１次元CNNをRNNの前処理ステップとして使用することで、RNN、で処理するための有益な表現を抽出するのが得策である