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

6.3 リカレントニューラルネットワークの高度な使い方

16T4063F 結城洸太

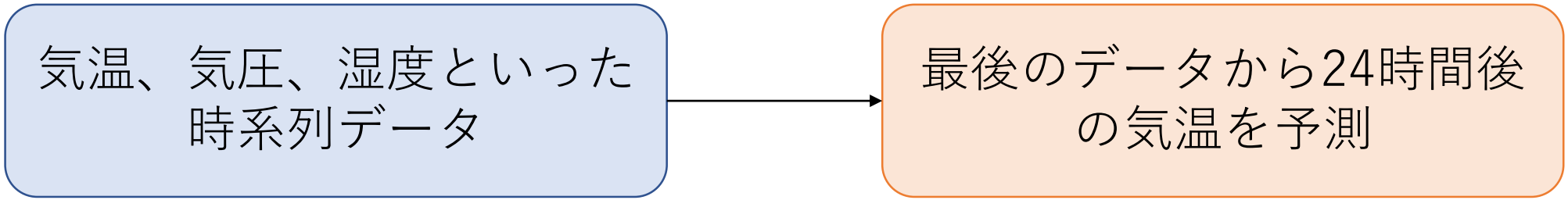
6.3 リカレントニューラルネットワークの高度な使い方

RNNの性能、汎化力を向上させる高度な手法

- リカレントドロップアウト
- リカレント層のスタッキング
- 双方向のリカレント層

気象予測問題を例に3つの手法を見ていく。

6.3.1 気象予想問題



ドイツのイエナにあるMax Planck Institute for Biogeochemistryの観測所で記録されたデータ(2009～2016年)を使用

- 14種類の数値(気温、気圧、湿度、風向など)を10分おきに記録した数年分のデータ

6.3.2 データの準備

パラメータの設定

- 過去**5日分**の観測データから、
- **1時間あたり1データ**の割合でサンプリングを行い、
- **24時間後**の気温を予測する。

準備

- データの前処理を行い、ネットワークに読み込めるフォーマットにする
- Pythonジェネレータを記述する。

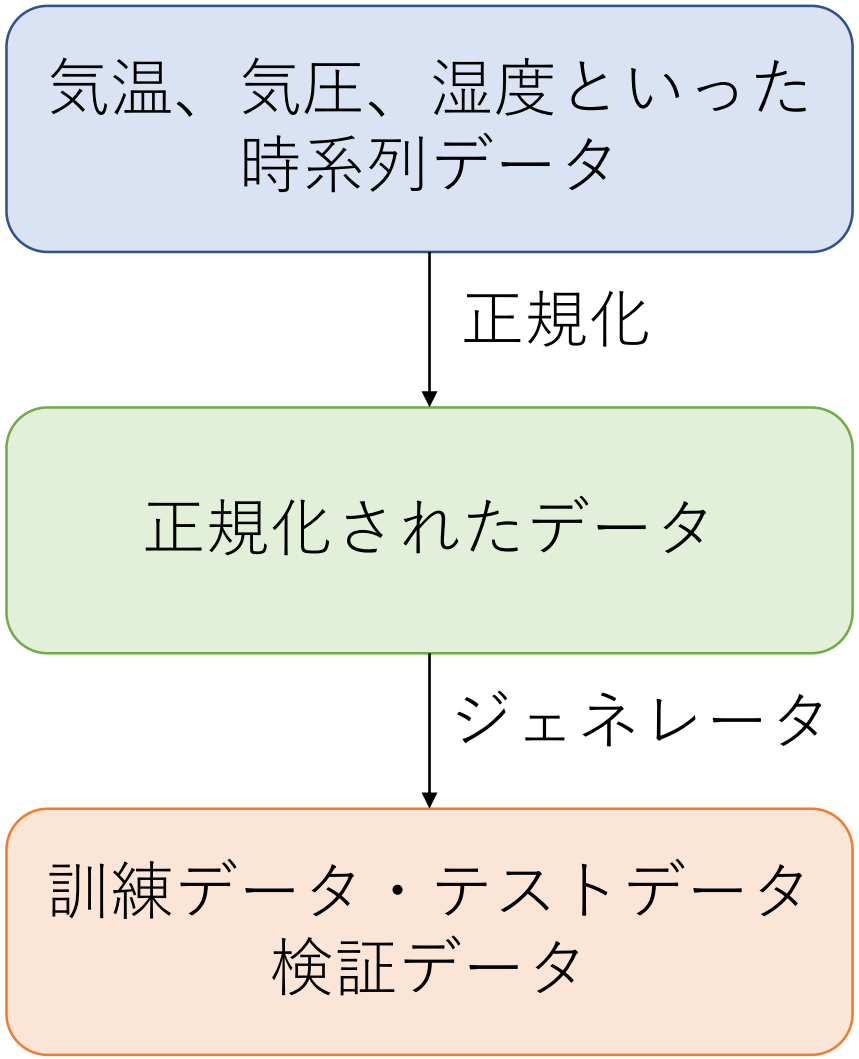
データの前処理を行い、ネットワークに読み込める形式にする

例えば、

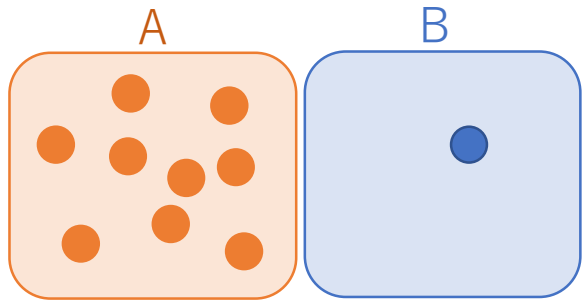
気温 … -20～+30(摂氏温度)
気圧 … 1,000前後(ミリバール単位)

全ての時系列が同じような尺度の小さな値になるように正規化する。
(平均を引いて、標準偏差で割る)

Pythonジェネレータを記述する



6.3.3 機械学習とは別の、常識的なベースライン



常識的なアプローチ

例　不均衡な分類問題（クラスAが90%、クラスBが10%）
常に「A」と予測する。→90％の正解率
機械学習でのアプローチが有益であることを実証するには、
「90%の正解率」というベースラインをクリアしなければならない。

気象予測問題における常識的なアプローチ

→常に、24時間後の気温が現在の気温と同じになると予測

その場合、

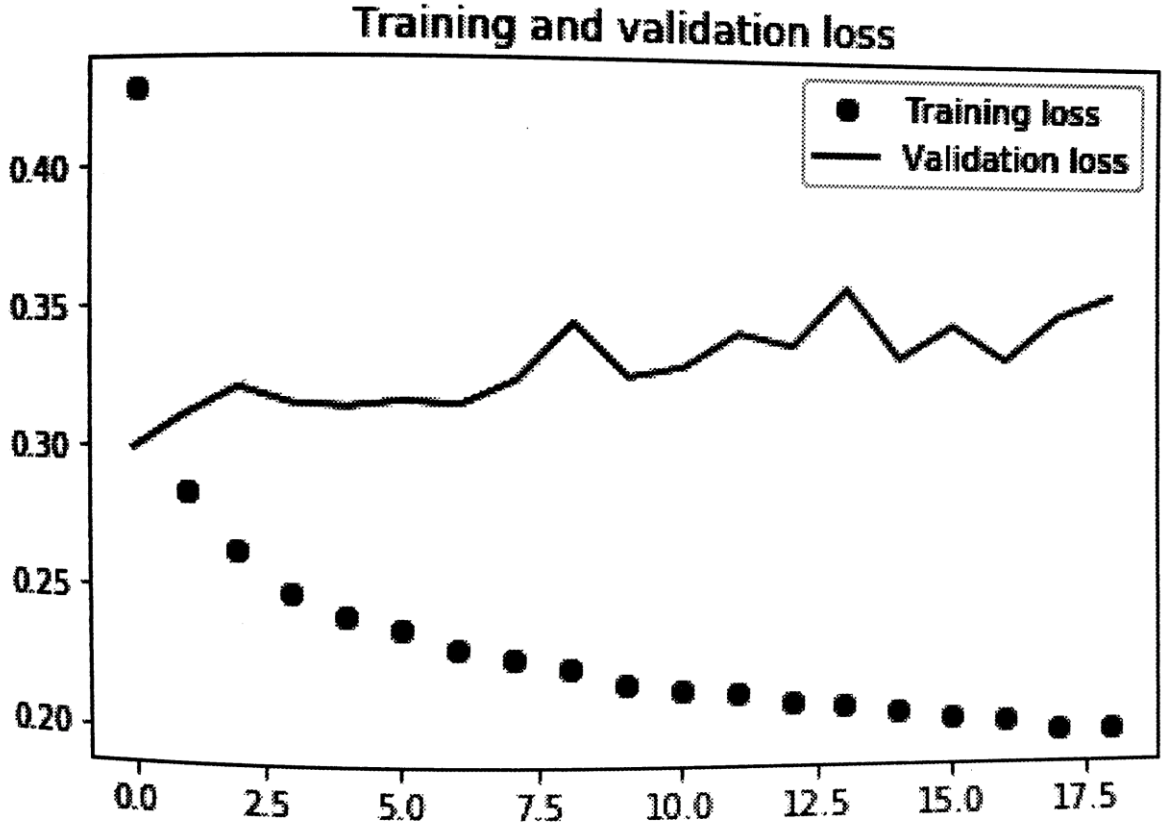
平均絶対誤差(MAE)=0.29　摂氏の誤差0.29×標準偏差=**2.57℃**

このベースラインより、良い結果を目指す。

6.3.4 機械学習の基本的なアプローチ

RNNの有益性を調べるために、全結合のモデルで試してみる。
損失関数には、常識的なアプローチと同じ平均絶対誤差(MAE)を使用。

(ドットが訓練データ、折れ線が検証データでの結果)
MAEは常識的なアプローチの0.29に近づいているものの、
確実ではない。



6.3.5 最初のリカレントベースライン

RNNでやってみる。

ここでは、前節で紹介したLSTM層の代わりにGRU層を使用。

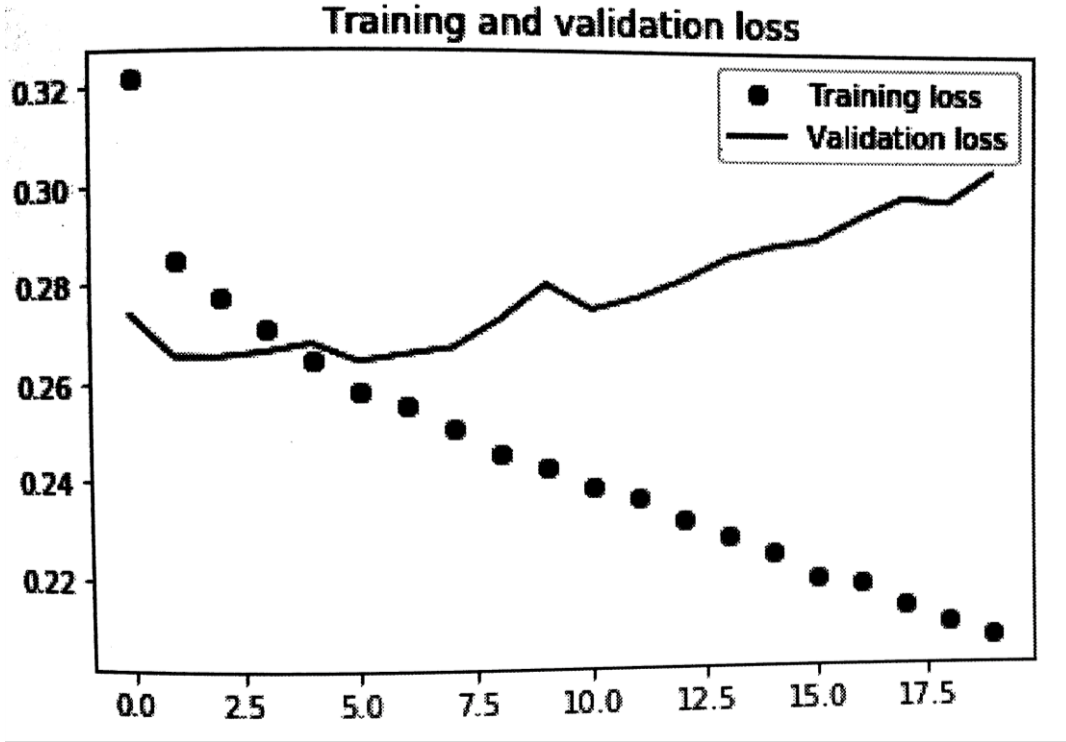
GRU層は、LSTMと同じ原理に基づいているが、少し効率化されていて実行コストがかからない。その分表現力は劣る。

平均絶対誤差は最小で0.265

非正規化後　2.35°C

最初の2.57°Cより前進した。

まだ改善の余地あり(過学習)

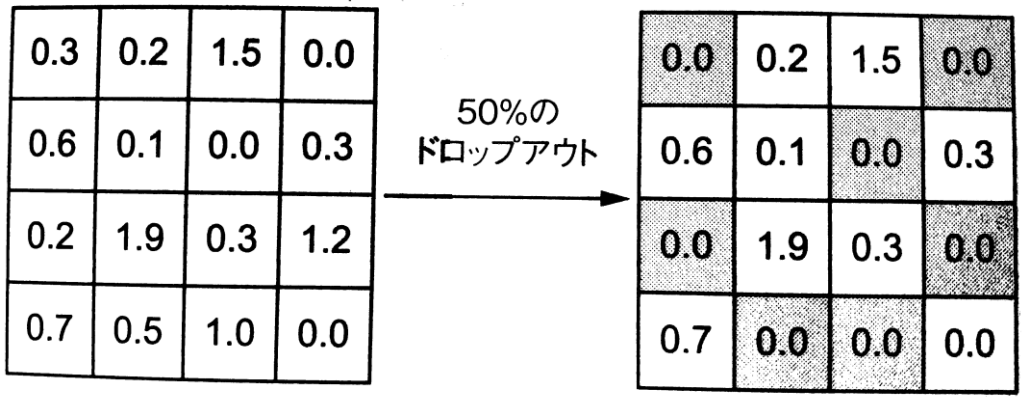


6.3.6 リカレントドロップアウトを使って過学習を抑制する

過学習を抑える手法

- ドロップアウト

RNNではリカレント層の前にドロップアウトを適応すると、学習の妨げになる。



RNNでドロップアウトを使用するための正しい方法

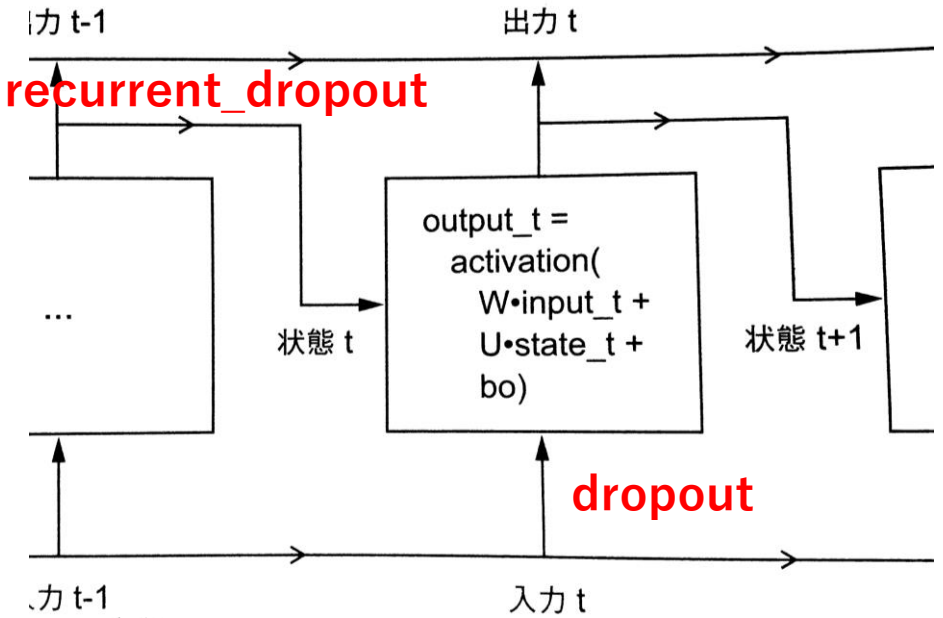
- 時間刻みごとにドロップアウトマスクをランダムに変化させるのではなく、すべての時間刻みで同じドロップアウトマスクを適応する。
- 時間的に一定のドロップアウトマスクを層の内部のリカレント活性化に適用する。

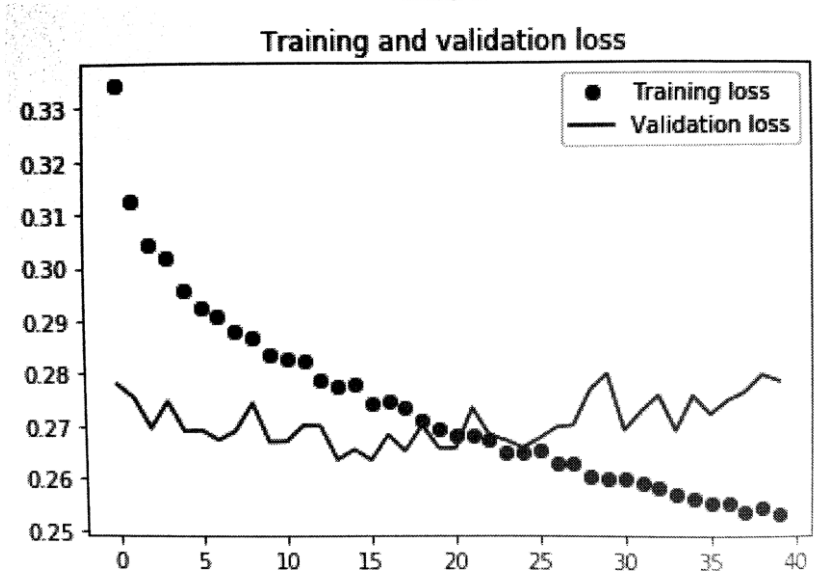
6.3.6 リカレントドロップアウトを使って過学習を抑制する

Kerasのリカレント層では、
dropoutとrecurrent_dropoutの２つのパラメータが定義されている。

ドロップアウトを使用する場合は、収束するのに時間がかかるため、エポック数を2倍にする。

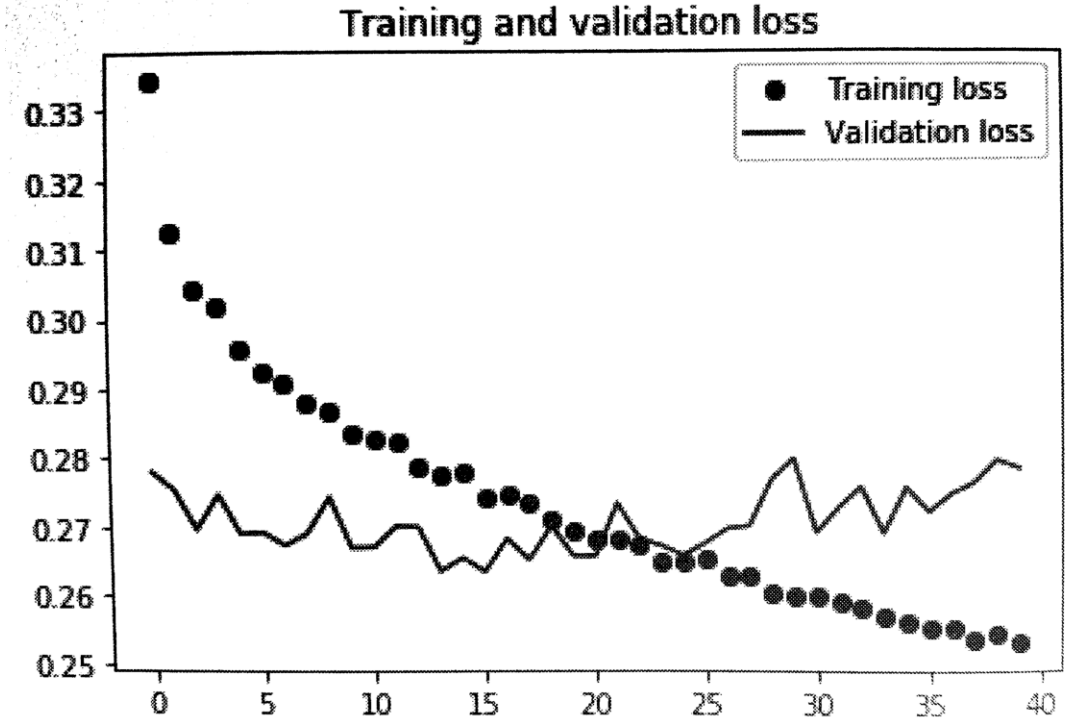
結果は、過学習がなくなり安定しているものの、ベストスコアは前より低くない。





6.3.7 リカレント層のスタッキング

過学習はなくなったので性能を改善する。
ネットワークのキャパシティ(層のユニット数か層の数)を増やしてみる。
ここでは、リカレント層をスタックとして積み上げる。

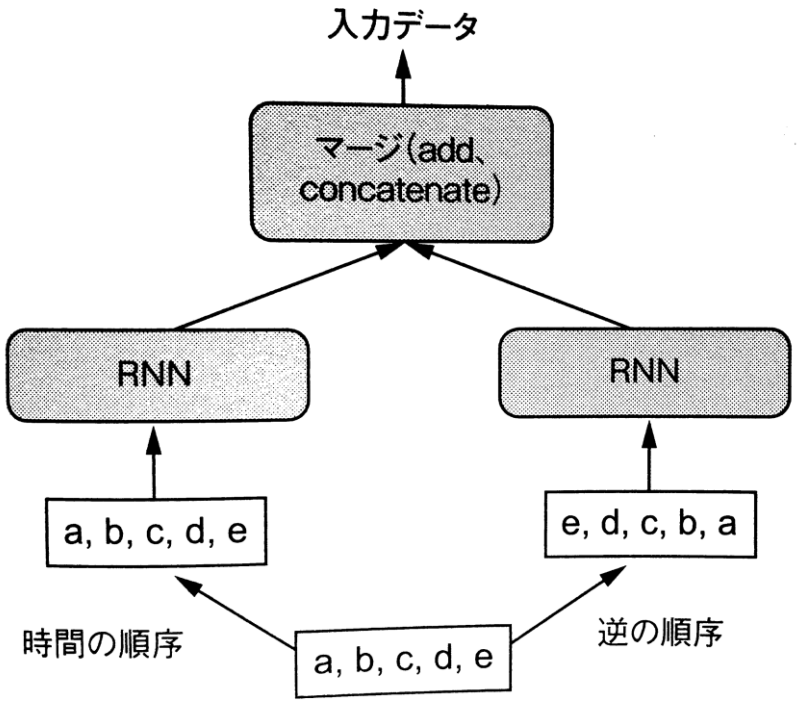


少し改善したが、それほど大きくない。

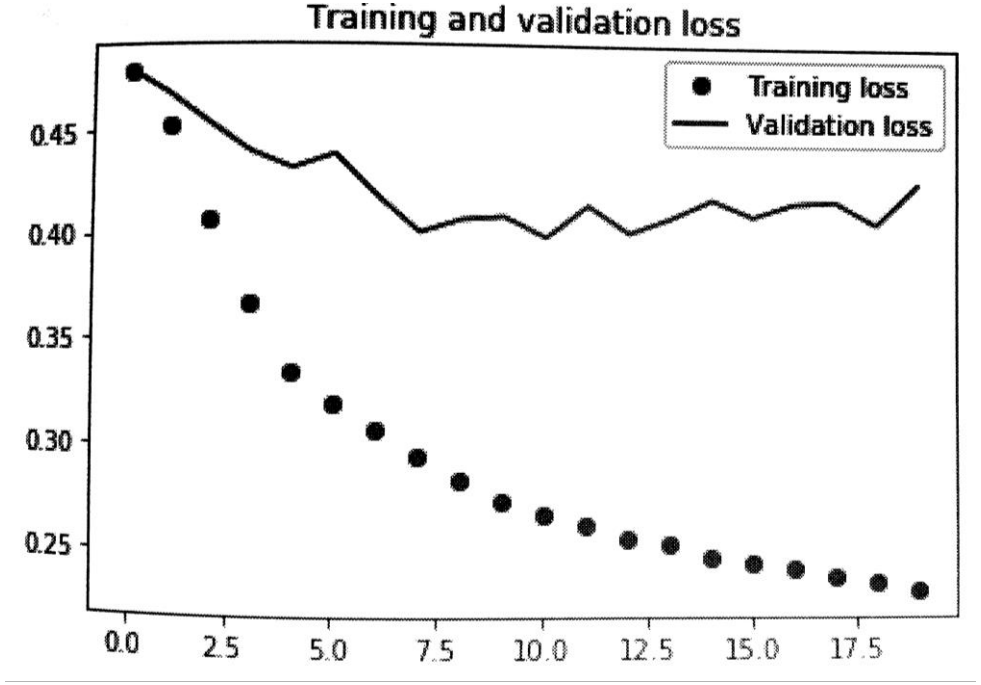
- 過学習がひどくないため、層のサイズを大きくしても安全。
- 層の追加で改善に至らなかったので、キャパシティを増やすことによる収穫逓減とみなせる。

6.3.8 双方向のRNN

双方向RNN



時間刻みの古いものから順にやってみる。



性能はかなり悪い。
時間の順序で処理することが重要だとわかる。

6.3.9 さらに先へ進むために

性能向上の方法いろいろ

- スタッキング設定において各リカレント層のユニットを調整する。現在の選択は根拠に乏しく、おそらく最適ではない。
- RMSpropオプティマイザが使用する学習率を調整する。
- GRU層の代わりにLSTM層を試してみる。
- より大きなDense層か、Dence層のスタックを試してみる。

6.3.10 まとめ

- 新しい問題に取り組むときは、ベースラインの設定が効果的。

- 複雑なモデルの正当化のため、単純なモデルも試してみる。

- 時間の順序が重要となるデータを使うときは、RNNが最適。

- RNNでのドロップアウト
- スタッキングRNN
- 双方向RNN