6.2　リカレントニューラルネットワークを理解する

17t4055g　築地俊平

6.2　リカレントニューラルネットワークを理解する

- 全結合ネットワークやCNN

→記憶を持っていない

シーケンス全体を一度にネットワークに提供することで単一のデータ点として扱う

→フィードフォワードネットワーク(feedforward network)

6.2　リカレントニューラルネットワークを理解する

人が文章を読むときは単語を記憶し、意味が流れる
→過去の情報から構築され、新しい情報が与えられるたびに更新

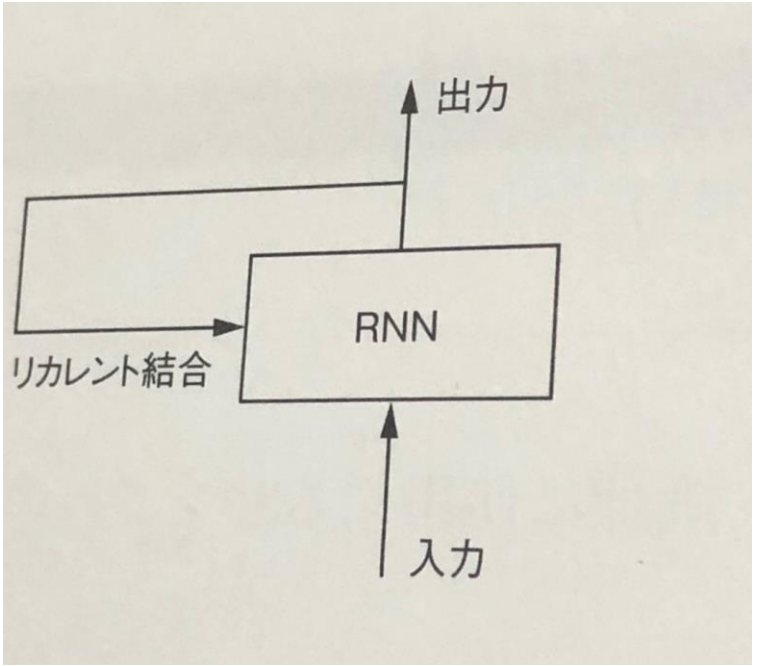
- **リカレントニューラルネットワーク(RNN)**

→シーケンスの要素を反復的に処理
→その過程で検出されたものに関連する情報は**状態**として維持
→内部ループを持つニューラルネットワークの一種

6.2　リカレントニューラルネットワークを理解する

- リカレントネットワークの疑似コード

```
state_t = 0 #時間tでの状態
for input_t in input_sequence:
    output_t = f(input_t,state_t)
    state_t = output_t  #一つ前の出力が次の状態
```



6.2　リカレントニューラルネットワークを理解する

- 単純な実装

```
import numpy as np

timesteps = 100
input_feature = 32
output_feature = 64

inputs = np.random.random((timesteps,input_feature))

state_t = np.zeros((output_feature,))

W = np.random.random((output_feature,input_feature))
U = np.random.random((output_feature,output_feature))
b = np.random.random((output_feature,))

successive_outputs = []

for input_t in inputs:
    output_t = np.tanh((np.dot(W,input_t)+np.dot(U,state_t)+b))
    successive_outputs.append(output_t)
    staet_t = output_t

final_output_sequence = np.stack(successive_outputs,axis=0)
```

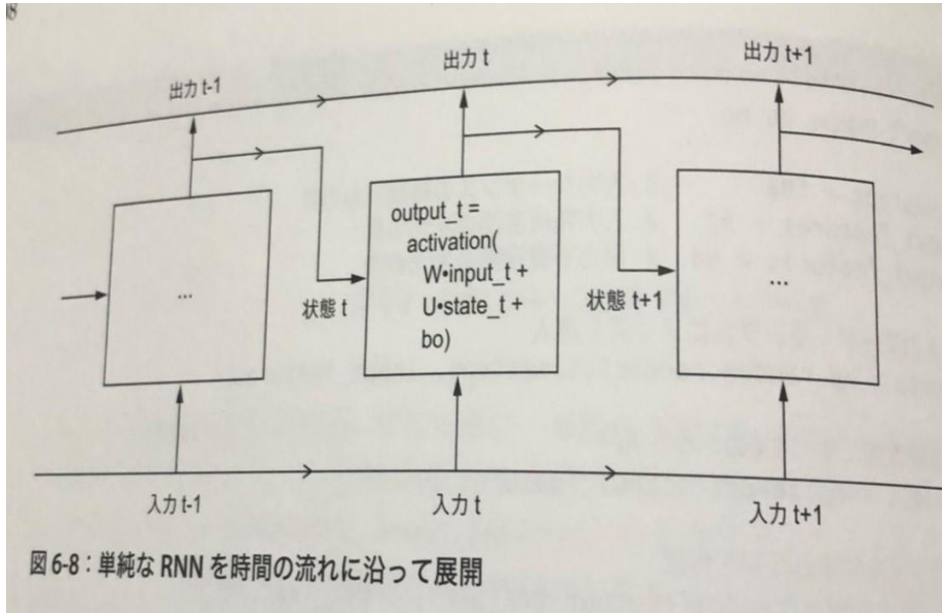


図 6-8：単純な RNN を時間の流れに沿って展開

6.2.1 Kerasでのリカレント層

- Numpyで実装した層はKerasではSimpleRNNで実装できる

※Numpy→実装での処理するシーケンスは一つ
　SimpleRNN→シーケンスのバッチを処理

6.2.1 kerasでのリカレント層

最後の時間刻みの出力

```
>>> from keras.layers import Embedding,SimpleRNN
>>> model = Sequential()
>>> model.add(Embedding(10000,32))
>>> model.add(SimpleRNN(32))
>>> model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
```

6.2.1 kerasでのリカレント層

- 完全な状態
- >>> model = Sequential()
- >>> model.add(Embedding(10000,32))
- >>> model.add(SimpleRNN(32,return_sequences=True))
- >>> model.summary()
- Model: "sequential_2"
- ____
- Layer (type) Output Shape Param #
- ===
- embedding_2 (Embedding) (None, None, 32) 320000
- ____
- simple_rnn_2 (SimpleRNN) (None, None, 32) 2080
- ===
- Total params: 322,080
- Trainable params: 322,080
- Non-trainable params: 0
- ____

6.2.1 kerasでのリカレント層

複数のリカレント層を順番に積み重ねていくと、表現力が高まる

```
>>> model = Sequential()
>>> model.add(Embedding(10000,32))
>>> model.add(SimpleRNN(32,return_sequences=True))
>>> model.add(SimpleRNN(32,return_sequences=True))
>>> model.add(SimpleRNN(32,return_sequences=True))
>>> model.add(SimpleRNN(32))
>>> model.summary()
```

6.2.1 kerasでのリカレント層

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_4 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_5 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_6 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
_________________________________________________________________
```

6.2.1 kerasでのリカレント層

IMDb映画レビュー分類問題に適用

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

max_features = 10000
max_len = 500
batch_size = 32

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

print('Pad sequences(sample x time)')
input_train = sequence.pad_sequences(input_train,maxlen=max_len)
input_test = sequence.pad_sequences(input_test,maxlen=max_len)
print('input_train shape:',input_train.shape)
print('input_test shapq:',input_test.shape)
```

```
from keras.models import Sequential
from keras.layers import Embedding,SimpleRNN,Dence

model = Sequential()
model.add(Embedding(max_features,32))
model.add(SimpleRNN(32))
model.add(Dence(1,activation='sugmoid'))

model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
history =model.fit(input_train,y_train,epochs=10,batch_size=128,validation_split=0.2)
```

6.2.1 kerasでのリカレント層

訓練データと検証データでの正解率と損失関数をプロット

```
import matplotlib.pyplot as plt

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

epochs = range(len(acc))

plt.plot(epochs,acc,'do',label='Training acc')
plt.plot(epochs,val_acc,'b',label='Validation acc')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

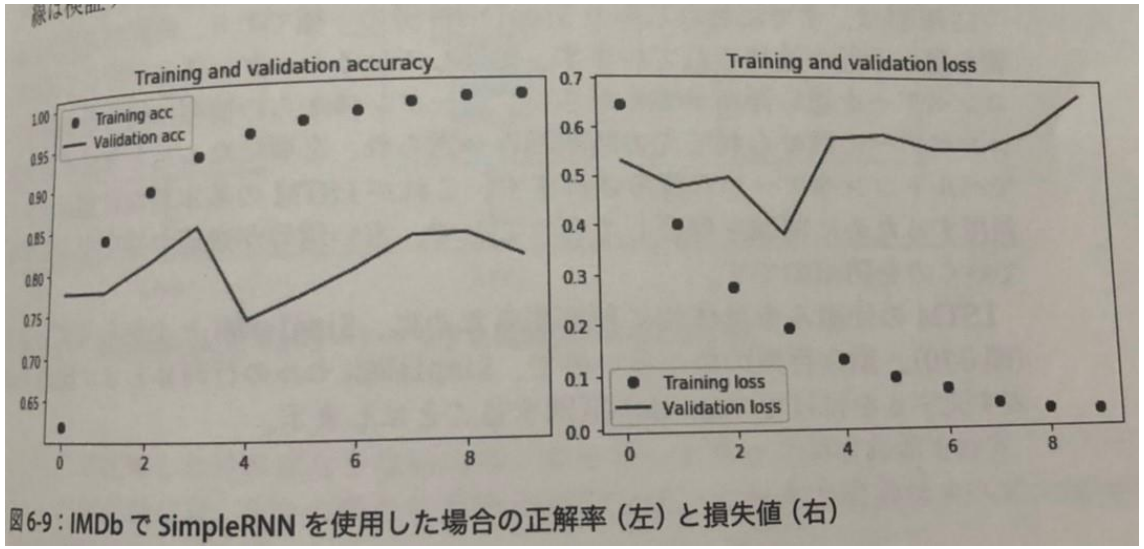


図6-9：IMDbでSimpleRNNを使用した場合の正解率（左）と損失値（右）

3章88％の正解率
今回→85%の正解率

長いシーケンスの処理に適していない

6.2.2　LSTM層とGRU層

Kerasのリカレント層はSimpleRNNだけでない

→LSTMとGRUが使用できる

SimpleRNN

→時間刻みで検出された入力に関する情報を維持できるはずが、長時間の依存関係を学習できない

→勾配消失問題

6.2.2　LSTM層とGRU層

LSTM層
→SimpleRNN層の1種であり、時間刻みにまたがって情報を運ぶ
→SimpleRNNにキャリートラックを追加したもの

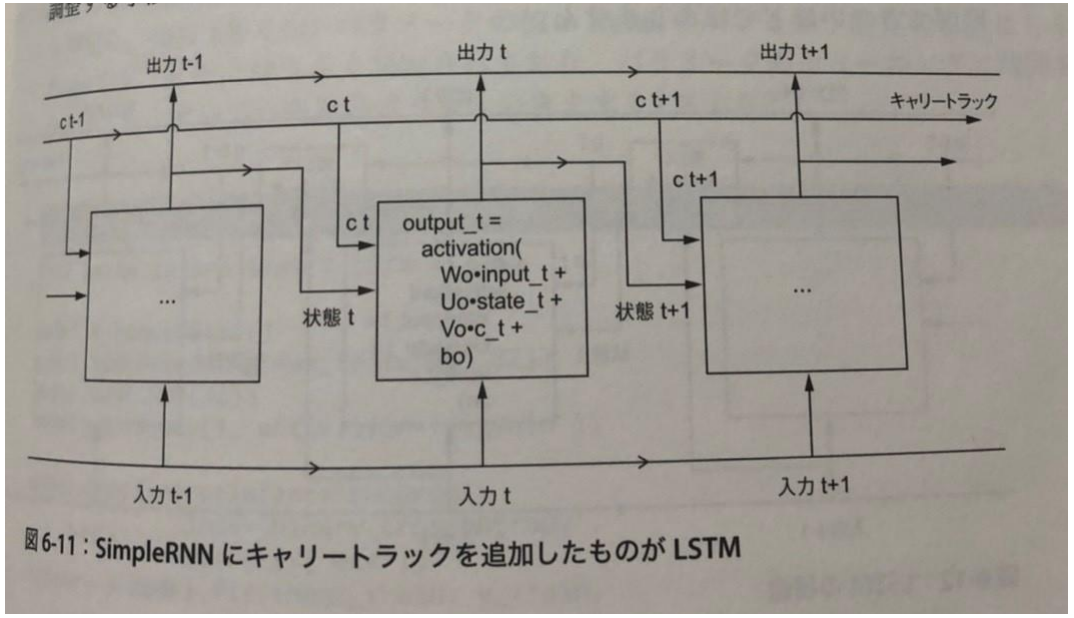


図 6-11：SimpleRNN にキャリートラックを追加したものが LSTM

6.2.3　kerasでのLSTMの具体的な例

LSTM層を使ってモデルを構築

```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_features,32))
model.add(LSTM(32))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['acc'])
history = model.fit(input_train,y_train,epochs=10,batch_size=128,validation_split=0.2)
```

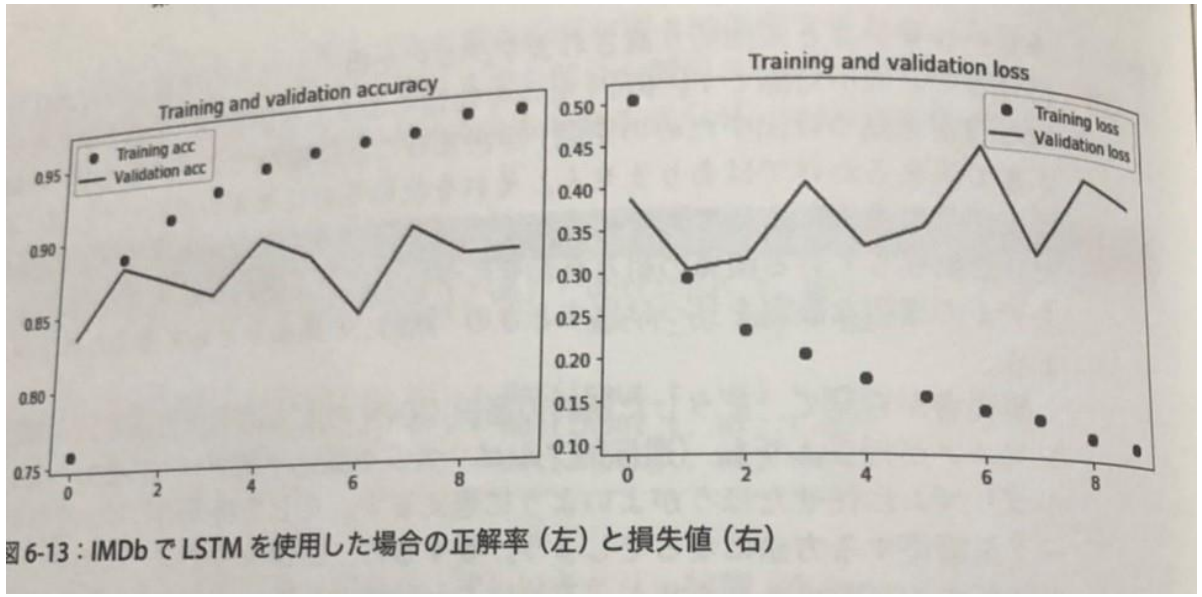


図 6-13：IMDb で LSTM を使用した場合の正解率（左）と損失値（右）

今回んお検証データで
の正解率は89％

6.2.3　kerasでのLSTMの具体的な例

SimpleRNNよりはよい結果

しかし、目を見張る結果とはいえない

→埋め込みの次元数やLSTMの出力の次元数といったハイパラメータのチューニングを行っていない

→正則化を行っていない

→映画レビューの大域的かつ長期的な構造の分析が、感情分析問題では役に立たない

→LSTMが役立つのは質疑応答や機械翻訳といった問題

6.2.4　まとめ

- RNNはどのようなものか
- LSTMはどのようなものか、長いシーケンスを単純なRNNよりもうまく処理するのはなぜか
- シーケンスデータを処理するためにKerasのRNN層をどのように使用するか