PythonとKerasによるディープラーニング

第七章：高度なディープラーニングのベストプラクティス

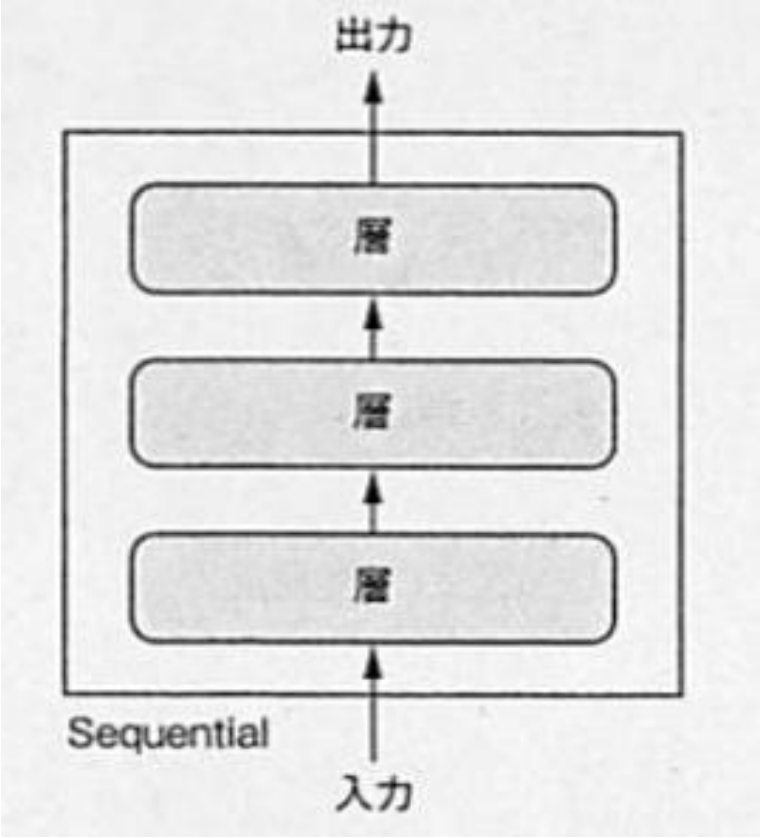
7.1　sequentialモデルを超えて：Keras　Functional　API

19ss317u　ZHAO YI

7.1　sequentialモデルを超えて：Keras Functional　API

- Sequentialモデル：

ネットワークの入力と出力がそれぞれ１つだけであることを前提として、層の線形スタックで構成される。



マルチモーダル入力

- マルチモーダルしたタスクでは、様々な入力ソースからのデータをマージし、様々な種類のニューラルネットワークを使って、各種のデータを処理する。
- 多入力例：古着の市場価格を予想するディープラニングモデルを思い浮かべてみる。
  モデル入力：

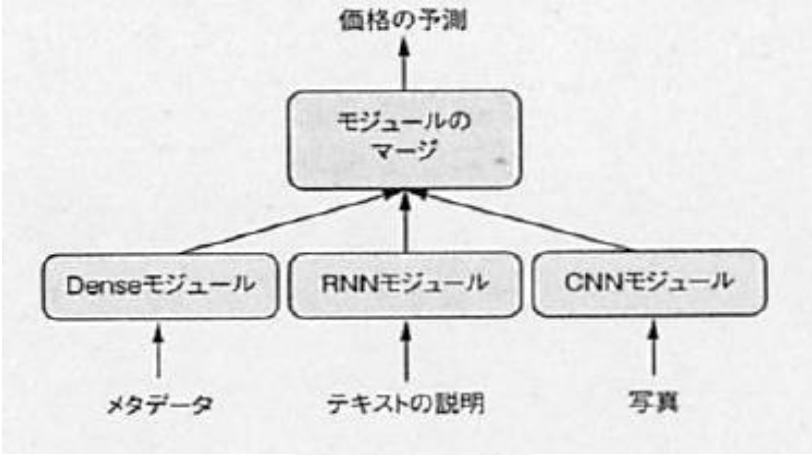
1.メタデータだけ：one-hotエンコーディングを実行し、全結合ネットワークを使って、価格を予測する。
2.テキストの説明だけ：リカレントニューラルネットワークか1次元畳み込みニューラルネットワーク(CNN)を利用できる。
3.商品の写真だけ：２次元のCNNを利用できる。

多入力の例

- 1,2,3同時に使用するとき：

　単純なアプローチ：３つのモデルを別々に訓練し、それらの予測値の荷重平均を求めることである。

　良い方法：利用可能なモーダル入力をすべて同時に参照できるモデルを使って、データからより性能のよいモデルを同時に学習することである。

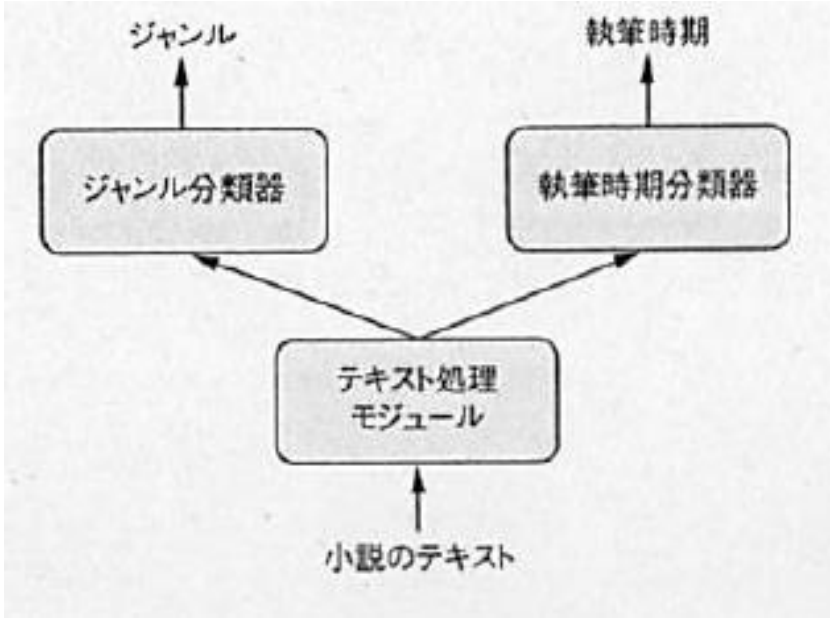


多入力モデル

多出力の例

- 長編小説や短編小説のテキストがあるとする。
　その小説がいつから執筆されたものであるかも予測したいと考えている。

予測面：1.モデルはジャンル
　　2.モデルは執筆時期



多出力（マルチヘッド）モデル

- Kerasでsequentialモデルクラスだけを使用する場合には不可能であるけれども、Kerasを使用するためにもう1つ柔軟な方法がある。

Functional　API

7.1.1　速習：Keras　Functional　API

- 概念：テンソルを直接操作し、テンソルを受け取ってテンソルを返す関数として層を使用する。

```
from keras import Input, layers

# テンソル
input_tensor = Input(shape=(32,))

# 層は関数
dense = layers.Dense(32, activation='relu')

# テンソルで呼び出された層はテンソルを返す
output_tensor = dense(input_tensor)
```

単純なSequentialモデルとFunctionalAPIを比較する。

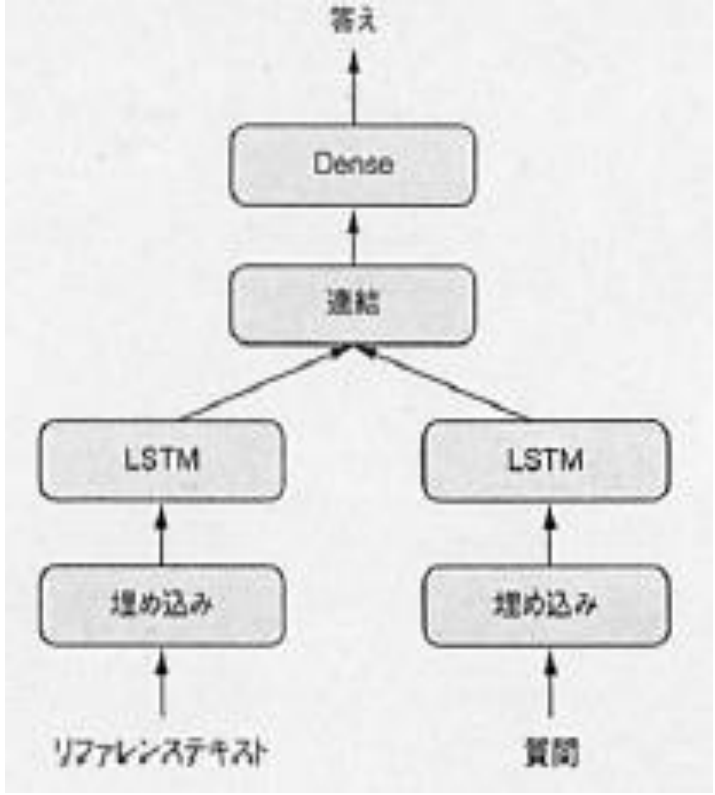
```
from keras.models import Sequential, Model
from keras import layers
from keras import Input

# すでにおなじみのSequentialモデル
seq_model = Sequential()
seq_model.add(layers.Dense(32, activation='relu', input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))

# Functional APIでそれに相当するもの
input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)

# Modelクラスは入力テンソルと出力テンソルをモデルに変換する
model = Model(input_tensor, output_tensor)

# このモデルのアーキテクチャを確認
model.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2080
dense_5 (Dense)	(None, 32)	1056
dense_6 (Dense)	(None, 10)	330

```
Total params: 3,466
Trainable params: 3,466
Non-trainable params: 0
```

- Kerasは、input_tensorからoutput_tensorまでの間にあろ層をすべて取得し、それらをグラフ形式のデータ構造、つまりモデルにまとめる。
- 無関係な入力と出力からモデルを構築した場合は、RuntimeErrorになる。

多入力モデルの単純な例：質問応答モデル



質問応答モデル

7.1.2多入力モデル

- リスト7-1は、FunctionalAPIを使って質問応答モデルを構築する例を示している。

　1)2つの独立した分岐を設定し、テキスト入力と質問入力を表現ベクトルとしてエンコードする。

　2)それらのベクトルを連結する。

　3)連結された表現の後にソフトマックス分類器を追加する。

リスト 7-1：2 つの入力を持つ質問応答モデルの Functional API 実装

```
from keras.models import Model
from keras import layers
from keras import Input

text_vocabulary_size = 10000
question_vocabulary_size = 10000
answer_vocabulary_size = 500

# テキスト入力は整数の可変長のシーケンス
# なお、必要であれば、入力に名前を付けることもできる
text_input = Input(shape=(None,), dtype='int32', name='text')

# 入力をサイズが64のベクトルシーケンスに埋め込む
embedded_text = layers.Embedding(
    text_vocabulary_size)(text_input, 64)

# LSTMを通じてこれらのベクトルを単一のベクトルにエンコード
encoded_text = layers.LSTM(32)(embedded_text)

# 質問入力でも（異なる層のインスタンスを使って）同じプロセスを繰り返す
question_input = Input(shape=(None,), dtype='int32', name='question')
embedded_question = layers.Embedding(
    question_vocabulary_size, 32)(question_input)
encoded_question = layers.LSTM(16)(embedded_question)

# エンコードされたテキストと質問を連結
concatenated = layers.concatenate([encoded_text, encoded_question],
                                  axis=-1)

# ソフトマックス分類器を追加
answer = layers.Dense(
    answer_vocabulary_size, activation='softmax')(concatenated)

# モデルをインスタンス化するときには、2つの入力と1つの出力を指定
model = Model([text_input, question_input], answer)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])
```

2つの入力を持つモデルをよく訓練する方法

- APIは2つある。

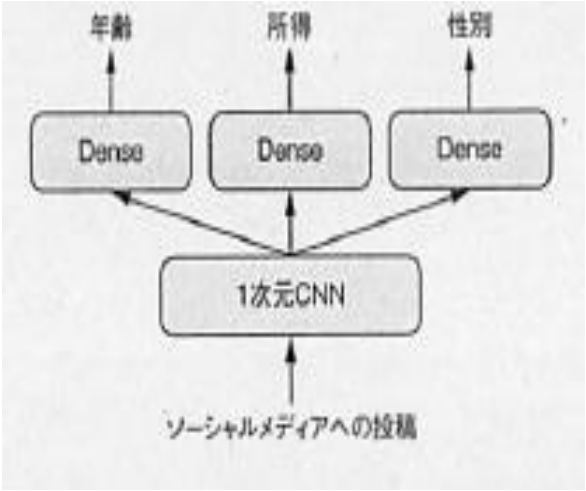
1)モデル入力としてnumpy配列のリストを供給できるAPI

2)入力の名前をnumpy配列にマッピングするディクショナリを供給できるAPI。

7.1.3　多出力モデル

- 例：

　同一のユーザーによるソーシャルメディアへの一連の投稿を入力として、そのユーザーの年齢、性別、所得といった複数の属性を予測しようとするネットワークが考えられる



3 つの出力 (ヘッド) を持つ
ソーシャルメディアモデル

モデルを訓練するにあたって、重要なこと

- ネットワークの様々なヘッドに対して異なる損失関数を指定できなければならないことである。
- 様々な損失値をまとめる簡単な方法：損失値の総和を求めること
- 損失値の貢献度がかなり不均衡である場合、モデルの表現は最も大きな損失値を持つタスクを優先する形で最適化されることになる。この問題をやり直すには、損失値に対して、最終的な損失値にどれぐらい貢献するのかを表す重要度を割り当てる。