PythonとKerasによるディープラーニング
第3章　入門：ニューラルネットワーク
3.4 二値分類の例：映画レビューの分類

16T4063F 結城洸太

3.4 二値分類の例：映画のレビューの分類

- 二値分類は、最も広く適用されている機械学習の一つ
- 映画のレビューのテキストの内容に基づいて、肯定的と否定的に分類する

3.4.1 IMDbデータセット

- IMDbデータセットを使用
  - IMDb(Internet Movie Database)から収集された、「肯定的」または「否定的」な50,000件のレビュー。
  - 訓練用25,000件、テスト用25,000件。それぞれ肯定的50%否定的50%

以下のコードでデータセットを読み込む。(リスト3-1)

```
#リスト3-1：IMDbデータセットの読み込み
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_word=10000)
```

3.4.2 データの準備

整数リストをテンソルに変換する方法

- リストをパディングしてすべて同じ長さに揃え、形状が(samples,word_indices)の整数型のテンソルに変更する。

  さらに、ネットワークの最初の層を、整数型のテンソルを扱える層として使用する。(埋め込み層)

- one-hotエンコーディングを使ってリストを0と1のベクトルに変更する。

(例)シーケンス[3,5]　→　[0., 0., 1., 0., 1., …0., 0., 0.](インデックス3と5が1、それ以外は0の10,000次元のベクトル)

そして、ネットワークの最初の層を、浮動小数点のベクトルデータを扱えるDence層として使用する。

3.4.2 データの準備

```
#リスト3-2：整数のシーケンスを二値行列に変換
import numpy as np

def vectorize_sequences(seqences, dimension=10000):
    #形状が(len(sequence),dimension)の行列を作成し、0で埋める
    results = np.zeros((len(sequence),dimension))

    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.    #results[i]のインデックスを1に設定
    return results

#訓練データのベクトル化
x_train = vectorize_sequences(train_data)
#テストデータのベクトル化
x_test = vectorize_sequences(test_data)
```

3.4.2 データの準備

- ラベルもベクトル化する

```
y_train = np.as array(train_labels).astype(‘float32’)
y_test = np.as array(test_labels).astype(‘float32’)
```

3.4.3 ニューラルネットワークの構築

- 入力データ＝ベクトル、ラベル＝スカラー(0,1)のように単純な設定の問題に適するのは、

単純な全結合層のスタックとReLU(Rectified Linear Unit)活性化関数で構築されたネットワーク。

3.4.3 ニューラルネットワークの構築

ReLU活性化関数を持つDense層の定義
Dense(16, activation='relu')
これは以下のテンソル演算の連鎖を実装する
output = relu(dot(W, input) + b)

引数16は隠れユニットの数。
=重み行列Wの形状が(input_dimension,16)になるということ。
入力データはWとの内積で16次元の表現空間に射影される。
(さらにバイアスベクトルbを足す)

3.4.3 ニューラルネットワークの構築

- 表現空間の次元は、ネットワークの自由度と考えられる。

隠れユニットの数が増える→　高次元の表現空間になる→
複雑な表現を学習できる
ただし、その分計算量が増える→
無駄なパターンを学習することにもなる。

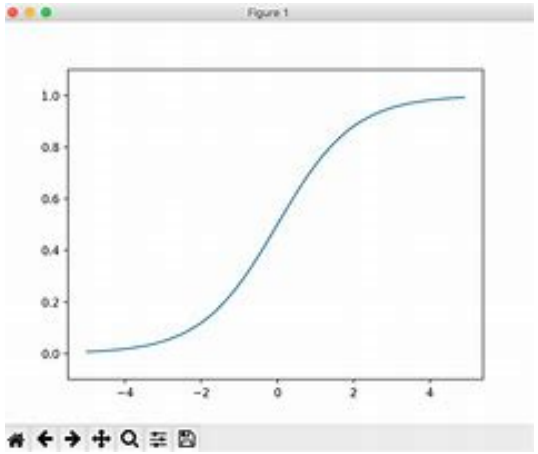
重要な決定

- 使用する層の数
- 各層の隠れユニットの数

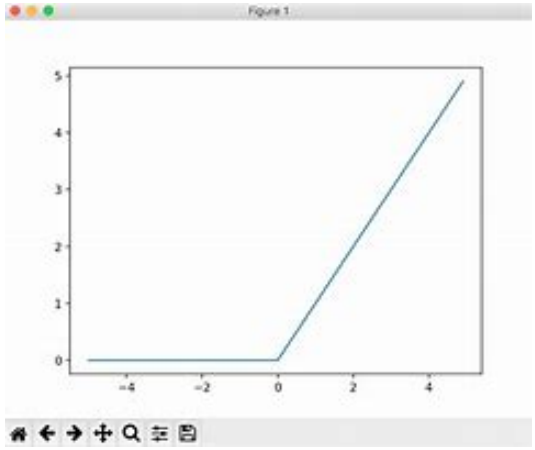
3.4.3 ニューラルネットワークの構築

今回は次のように選択する。

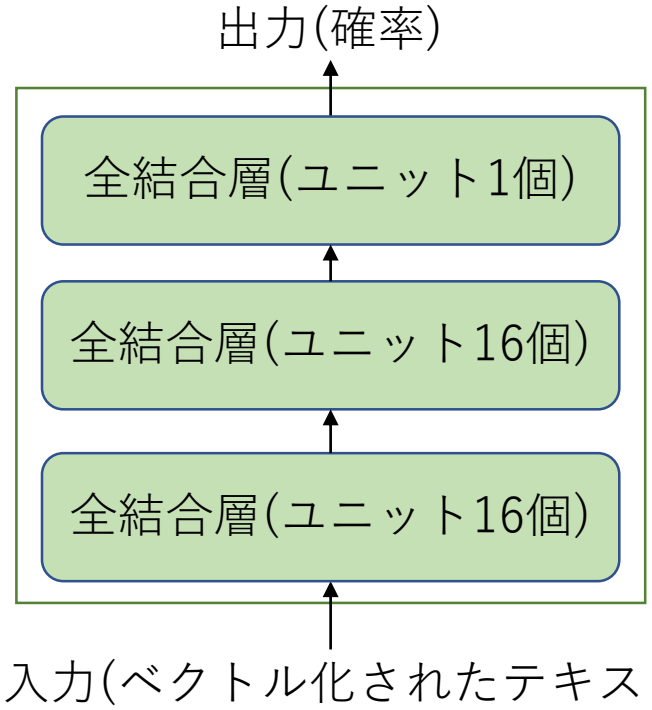
- それぞれ16個の隠れユニットを持つ2つの中間層(ReLU関数)
- 現在のレビューの感情に関する予測値を出力する3つ目の層(シグモイド関数)



シグモイド関数
全ての値を0〜1に
確立として解釈可



ReLU関数
負の数すべてを0に



3.4.3 ニューラルネットワークの構築

```
#リスト3-3モデルの定義
from keras import models
from keras import laters

model = models.Sequencial()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu',))
model.add(layers.Dense(1, activation='sigmoid' ))
```

3.4.3 ニューラルネットワークの構築

最後に、損失関数とオプティマイザを選択する。

二値問題なので、損失関数にはbinary_crossentropy(二値の交差エントロピー)を使用するのが効果的。

オプティマイザは、rmspropを使う。

```
#リスト3-4：モデルのコンパイル
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

3.4.4 アプローチの検証

全く新しいデータでモデルを訓練するときの正解率を監視する。

検証データセットを作成。

```
#リスト3-7：検証データセットの設定
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

3.4.4 アプローチの検証

x_trainテンソルとy_trainテンソルのすべてのサンプルで訓練を20回繰り返す。(20エポック)

同時に、10,000サンプルでの損失率と正解率を監視

検証データはvalidation_dataパラメータに引数指定

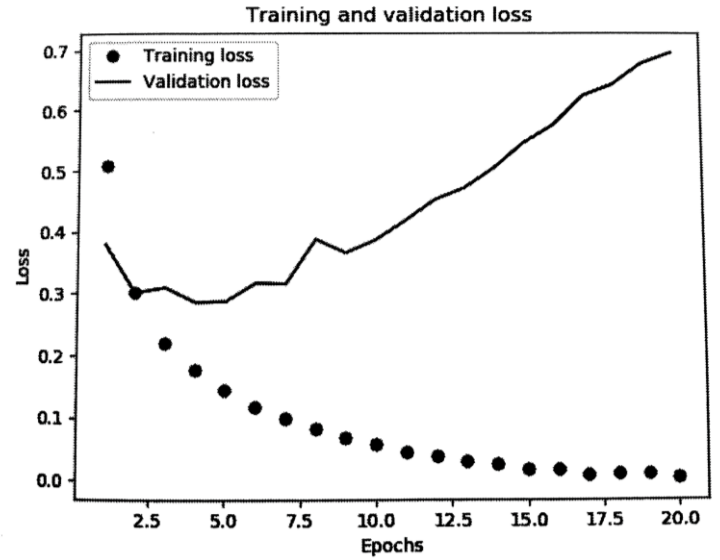
```
#リスト3-8：モデルの訓練
model.compile(optimizer='rmsprop',
                    loss='binary_crossentropy',
                   metrics=['acc']
history = model.fit(partial_x_train,
                        partial_y_train,
                        epochs = 20,
                        batch_size=512,
                        validation_data=(x_val, y_val))
```

3.4.4 アプローチの検証

右の図は、訓練データと検証データの損失率及び正解率をプロットしたもの

(ドット＝訓練データ、

折れ線＝検証データ)

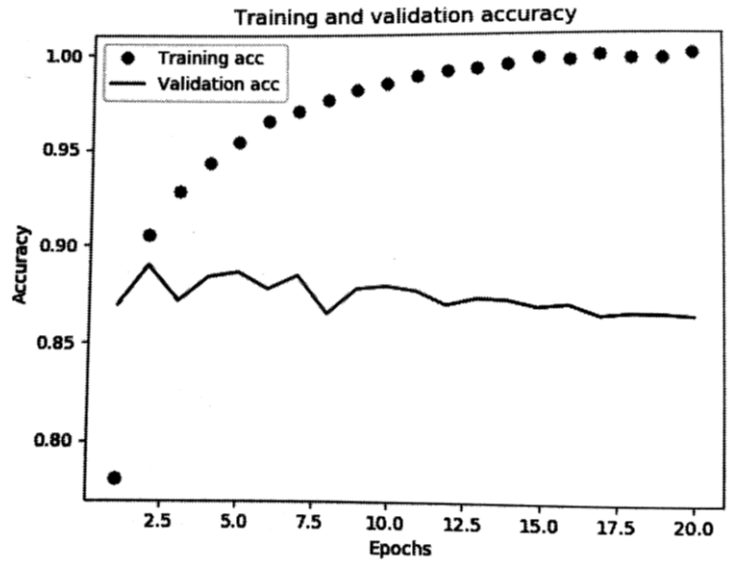


訓練データと検証データでの損失率

訓練データはエポックごとに、損失率は小さく、正解率は向上

検証データは4つ目のエポックがピーク

(過学習)



訓練データと検証データでの正解率

3.4.4 アプローチの検証

最終的な結果

```
>>>results
```

[0.29184698499679568, 0.88495999999999997]

88%の正解率を達成

3.4.4 アプローチの検証

過学習を回避するために4エポックで訓練してみる

```
#リスト3-11：モデルの訓練をやり直す
model = models.Sequencial()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu',))
model.add(layers.Dense(1, activation='sigmoid' ))

model.compile(optimizer='rmsprop',
                    loss='binary_crossentropy',
                   metrics=['accuracy']

model.fit(partial_x_train, partial_y_train, epochs = 4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

3.4.6 その他の実習

改善の余地

- 隠れ層を２つから、１つや３つなどに変更した場合に、正解率への影響を確認する。
- 隠れユニットの数を多く、または少なくしてみる。
- 損失関数をbinary_crossentropyからmseに変更してみる。
- 活性化関数をreluからtanhに変更してみる。

まとめ

本節で学んだこと

- 生のデータを前処理して、ネットワークに供給する。
- 活性化関数としてReLUを使用するDense層のスタックは幅広く使える。
- 二値分類問題では、最後の層にシグモイド関数を使用する。
- 上記の場合、損失関数はbinary_crossentropyを使用する。
- rmspropオプティマイザはどんな問題で用いてもよい。
- 過学習に陥るので注意が必要である。