PythonとKerasによるディープラーニング

3.4 二値分類の例：映画レビューの分類

17T4014Y 伊藤 陽樹

IMDbデータセット

- 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.
- 訓練用の25,000件のレビューとテスト用の25,000件のレビュー.
  - 否定的な50%のレビューと肯定的な50%のビュー.
- Kerasでは、レビューの内容(単語のシーケンス)は整数のシーケンスに変換されており、整数はそれぞれ辞書の特定の単語を指す.

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_words=10000)

>>> train_data[0]
[1, 14, 22, 16, … , 178, 32]
>>> train_labels[0]
1

>>> max([max(sequence) for sequence in train_data])
9999
```

データの準備

- 整数のリストはテンソルに変換して、NNに供給.
- 2つの方法
  1. リストをパディングしてすべて同じ長さに揃えた上で、形状が(sample, word_indices)の整数型のテンソルに変換.
  2. one-hotエンコーディングを使ってリストを0と1のベクトルに変換.

- ここでは、2つ目の方法を使ってデータをベクトル化する.

整数のシーケンスを二値行列に変換

```
import numpy as np

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

データの準備

- サンプル

```
>>> x_train[0]
array([0., 1., 1., ..., 0., 0., 0.])
```

- ラベルもベクトル化しておく.

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

ニューラルネットワークの構築

- 入力データはベクトル、ラベルはスカラー(1と0).
  - 単純な全結合層のスタックとReLU活性化関数で構成されたネットワーク.

    Dense(16, activation='relu')

- 引数(16)は、その層の隠れユニットの数.
- 隠れユニット(hidden unit)は、その層の表現空間において1つの次元を表す.

ニューラルネットワークの構築

- 隠れユニットの数が増え、より高次元の表現空間になるほど、ネットワークはより複雑な表現を学習できる.
- ただし、その分ネットワークの計算量が増え、無駄なパターンを学習してしまう.

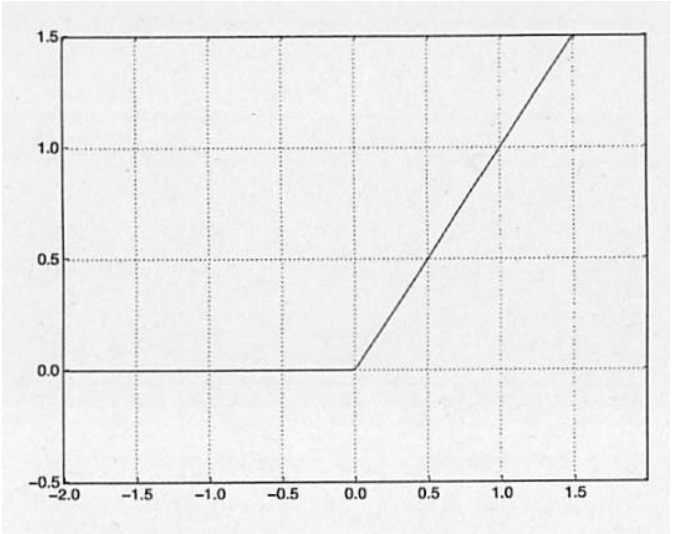
- Dense層のスタックでは、
  - ✓使用する層の数をいくつにするか
  - ✓各層の隠れユニットの数をいくつにするか

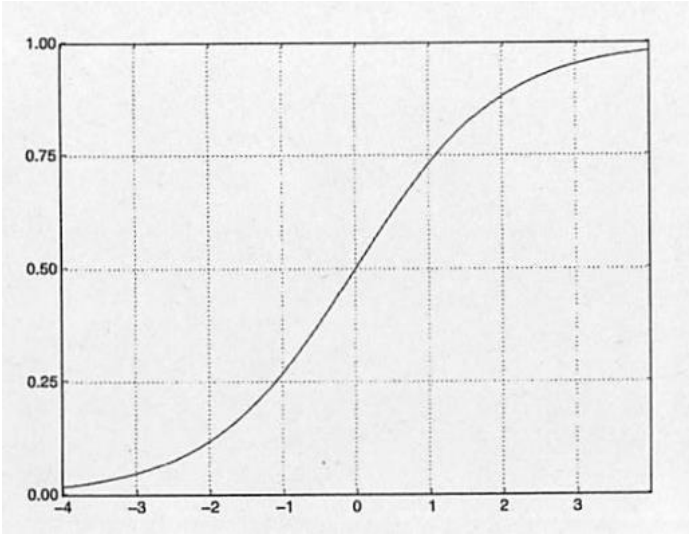
ここでは、

- ✓それぞれ16個の隠れユニットを持つ2つの中間層
- ✓現在のレビューの感情に関する予測値(スカラー)を出力する3つ目の層

- 2つの中間層では、活性化関数はReLU.
- 最後の3つ目の層では、シグモイド活性化関数.

ReLU





シグモイド

ニューラルネットワークの構築

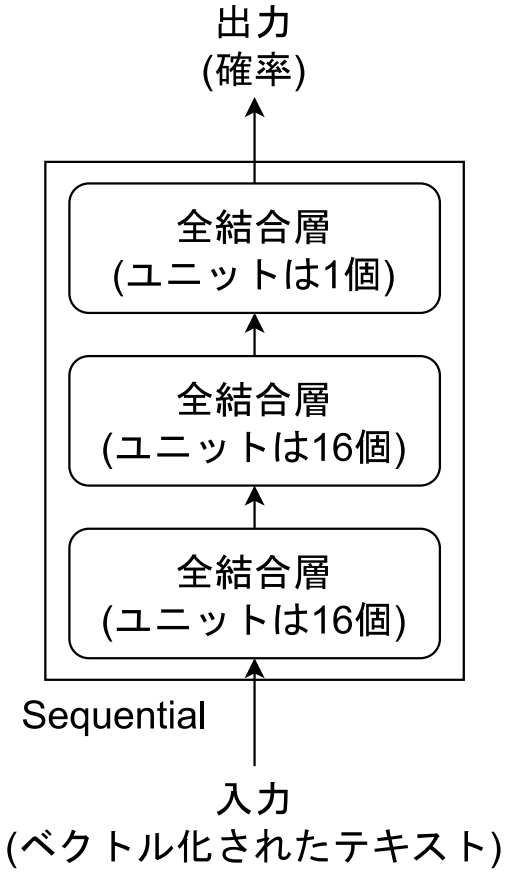


図: 3層のNN

- Kerasでの実装

```
from keras import models
from keras import layers

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

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の選択.
- 二値分類問題、ネットワークの出力は確率.
  - 損失関数は、binary_crossentropy(二値の交差エントロピー).
  - オプティマイザは、rmsprop

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

※訓練時に正解率も監視する

アプローチの検証

- 検証データセットの作成

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

- 512サンプルのミニバッチで20エポックの訓練

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

アプローチの検証

- Historyオブジェクトのhistoryメンバーは、訓練中に起きたすべてのことに関するデータを含んでいるディクショナリ.

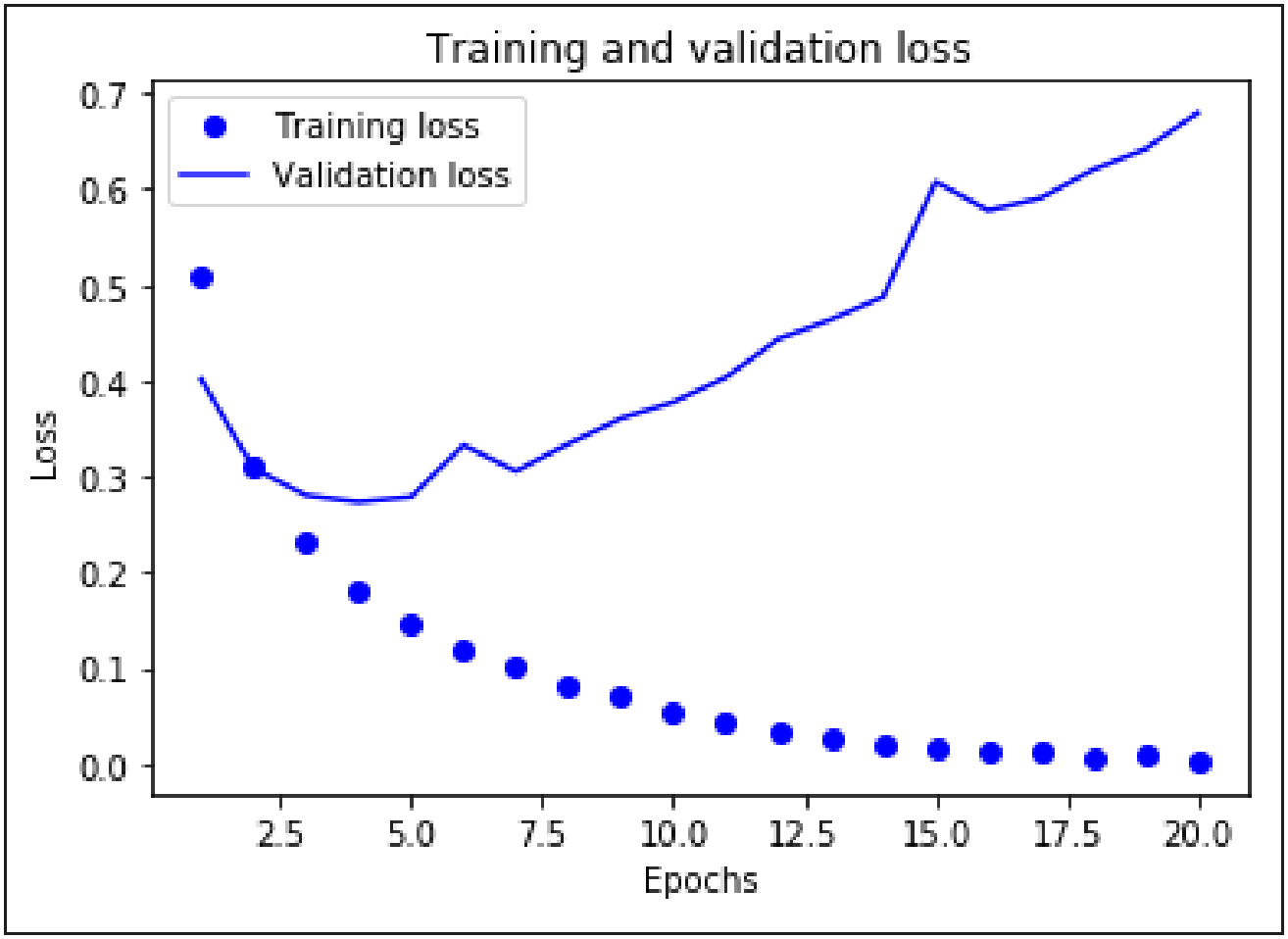
```
>>> history_dict = history.history
>>> history_dict.keys()
dict_keys(['val_acc', 'acc', 'val_loss', 'loss’])
```

- 訓練中及び検証中に監視される指標ごとに1つ、合計4つのエントリが含まれている.

訓練データと検証データでの損失値をプロット

```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict[‘loss’]
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)

# “bo”は”blue dot”(青のドット)を意味する
plt.plot(epochs, loss, 'bo', label='Training loss')

# “b”は”solid blue line”(青の実線)を意味する
plt.plot(epochs, val_loss, 'b', label='Validation loss’)

plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

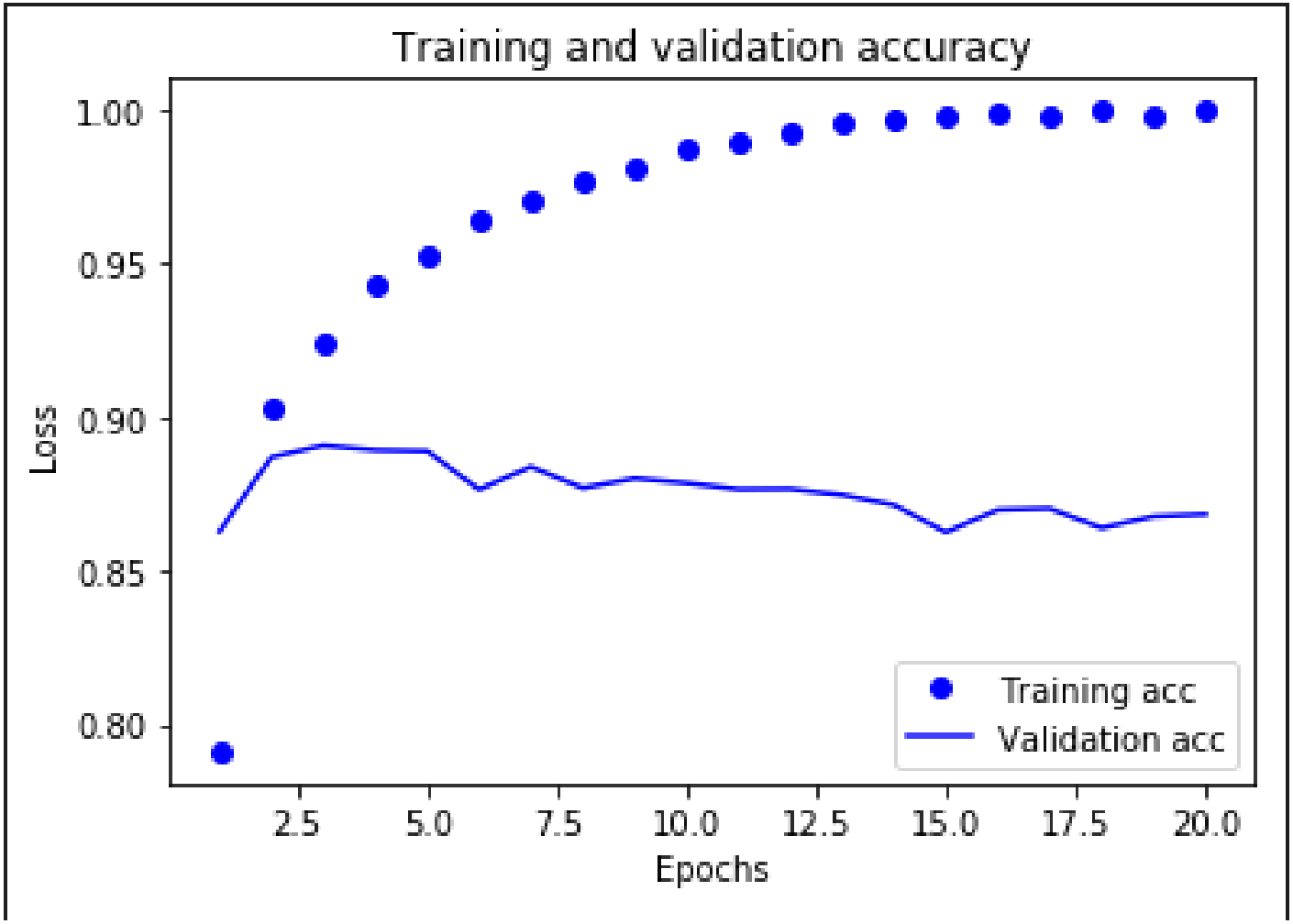
図: 訓練データと検証データでの損失値



訓練データと検証データでの正解率をプロット

```
plt.clf()  # 図を消去
acc = history_dict['acc']
val_acc = history_dict['val_acc']

plt.plot(epochs, accs, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel(‘Accuracy')
plt.legend()
plt.show()
```

図: 訓練データと検証データでの正解率



アプローチの検証

- 訓練データでの損失値がエポックごとに小さくなっている.
- 訓練データのデータの正解率がエポックごとに向上している.
  - 勾配降下法による最適化に期待される結果.

- 検証データでの損失値と正解率は、4つ目のエポックでピークに達している.
  - 過学習の一例

アプローチの検証

- 2つ目のエポックの後、このモデルは訓練データの過学習に陥っている.
  - 訓練を3エポックで中止する.

- 新しいモデルを4エポックで訓練し、テストデータで評価.

```
model = models.Sequential()
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(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

アプローチの検証

- 最終的な結果

```
>>> results
[0.29184698499679568, 0.8849599999999997]
```

- このアプローチでは、88%の正解率.
- 最先端のアプローチでは、95%近い正解率(らしい)

まとめ

- 通常、生のデータをNNに供給するには、データの前処理を行う.
- 活性化関数としてReLUを使用するDense層のスタックは、感情分類をはじめ、幅広い問題を解決できる.
- 二値分類問題(出力クラスが2つ)では、ネットワークの最後の層は、シグモイド関数を使用する単一ユニットのDense層. 出力は確率を表すスカラー値.
- 二値分類問題の出力がシグモイド関数のスカラー値の場合は、損失関数はbinary_crossentoropyを使用する.
- RMSpropオプティマイザは、一般にどのような問題でも十分によい選択.
- 訓練データでの性能がよくなっていくうちに、NNは過学習に陥るようになる.