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
3.5　多クラス分類の例：ニュース配信の分類

19nm715n ZHANGYIHANG

Reutersのニュース配信を46種類の相互排他なトピックに分類するネットワークを構築します。クラスの数が多いため、この問題は多クラス分類の一例です。

- 多クラス単一ラベル分類
  各データ点は１つのカテゴリにのみ分類されます。（Reuters問題）
- 多クラス多ラベル分類
  各データ点は複数のカテゴリに分類される可能性があります。

3.5.1　Reutersデータセット

　Reutersデータセットは、1986年にReutersによって配信された短いニュース記事とそれらのトピックを集めたものであり、テキスト分類用の単純なデータセットとして広く利用されています。トピックは全部で46種類です。

- Reutersデータセットを読み込む

```
In [1]: from keras.datasets import reuters
        Using TensorFlow backend.

In [2]: (train_data,train_labels),(test_data,test_labels)=reuters.load_data(num_words=10000)

In [3]: len(train_data)
Out[3]: 8982
```

10000個に制限

```
In [4]: len(test_data)
Out[4]: 2246

In [5]: train_data[10]
Out[5]: [1,
         245,
         273,
         207,
         156,
```

各サンプルは整数のリスト

- ニュースサンプルをテキストに変換する

```
In [8]: word_index=reuters.get_word_index()
        reverse_word_index=dict([(value,key) for (key,value) in word_index.items()])
        decoded_newswire=' '.join([reverse_word_index.get(i-3,'?') for i in train_data[0]])
        decoded_newswire
Out[8]: '? ? ? said as a result of its december acquisition of space co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs per share up from
        70 cts in 1986 the company said pretax net should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and rental operation revenues to 19
        to 22 mln dlrs from 12 5 mln dlrs it said cash flow per share this year should be 2 50 to three dlrs reuter 3'

In [9]: train_labels[10]
Out[9]: 3
```

3.5.2　データの準備

- データのエンコーディング

In [15]:
```
import numpy as np

def vectorize_sequences(sequences,dimension=10000):
    results = np.zeros((len(sequences),dimension))
    for i,sequence in enumerate(sequences):
        results[i,sequence]=1.
    return results
x_train=vectorize_sequences(train_data)
x_test=vectorize_sequences(test_data)
```

データのベクトル化

In [16]:
```
def to_one_hot(labels,dimension=46):
    results = np.zeros((len(labels),dimension))
    for i,label in enumerate(labels):
        results[i,label]=1.
    return results
one_hot_train_labels=to_one_hot(train_labels)
one_hot_test_labels=to_one_hot(test_labels)
```

One-hotエンコーディングを使ってラベルはそれぞれベクトルとして埋め込まれます

ラベルのインデックスの位置に１、以外はすべで０

In [17]:
```
from keras.utils.np_utils import to_categorical
one_hot_train_labels=to_categorical(train_labels)
one_hot_test_labels=to_categorical(test_labels)
```

MNISTの例ですでに確認したように、Kerasには、同じことを行う方法がすでに組み込まれています。

3.5.3　ニューラルネットワークの構築

- モデルの定義

In [21]:
```
from keras import models
from keras import layers

model=models.Sequential()
model.add(layers.Dense(64,activation='relu',input_shape=(10000,)))
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(46,activation='softmax'))

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

情報が失われないように、64ユニットの中間層で試します

出力は46種類の出力クラスの確率分布

最適な損失関数です。2つの確率分布の距離を計測します。

3.5.4　アプローチの検証

- 検証データセットの設定

In [22]:
```
x_val=x_train[:1000]
partial_x_train=x_train[1000:]

y_val=one_hot_train_labels[:1000]
partial_y_train=one_hot_train_labels[1000:]
```

- モデルの訓練

In [23]:
```
history=model.fit(partial_x_train,partial_y_train,epochs=20,batch_size=512,
                  validation_data=(x_val,y_val))
```

```
55 - val_loss: 0.9661 - val_acc: 0.8060
Epoch 15/20
7982/7982 [==============================] - 4s 470us/step - loss: 0.1392 - acc: 0.95
60 - val_loss: 0.9709 - val_acc: 0.8150
Epoch 16/20
7982/7982 [==============================] - 4s 478us/step - loss: 0.1317 - acc: 0.95
64 - val_loss: 1.0270 - val_acc: 0.8040
Epoch 17/20
7982/7982 [==============================] - 4s 463us/step - loss: 0.1220 - acc: 0.95
79 - val_loss: 1.0247 - val_acc: 0.7990
Epoch 18/20
7982/7982 [==============================] - 4s 445us/step - loss: 0.1201 - acc: 0.95
78 - val_loss: 1.0450 - val_acc: 0.8050
Epoch 19/20
7982/7982 [==============================] - 4s 464us/step - loss: 0.1141 - acc: 0.95
94 - val_loss: 1.1005 - val_acc: 0.7950
Epoch 20/20
7982/7982 [==============================] - 4s 466us/step - loss: 0.1113 - acc: 0.95
97 - val_loss: 1.0739 - val_acc: 0.8000
```

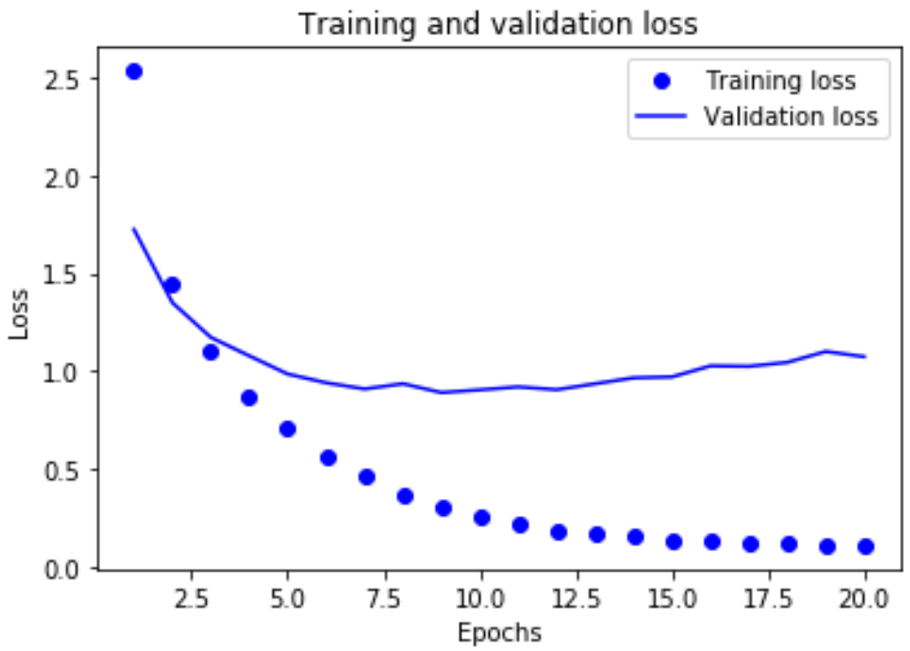
- 訓練データと検証データでの損失値をプロット

In [26]:
```
import matplotlib.pyplot as plt

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

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

plt.plot(epochs,loss,'bo',label='Training loss')
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()
```

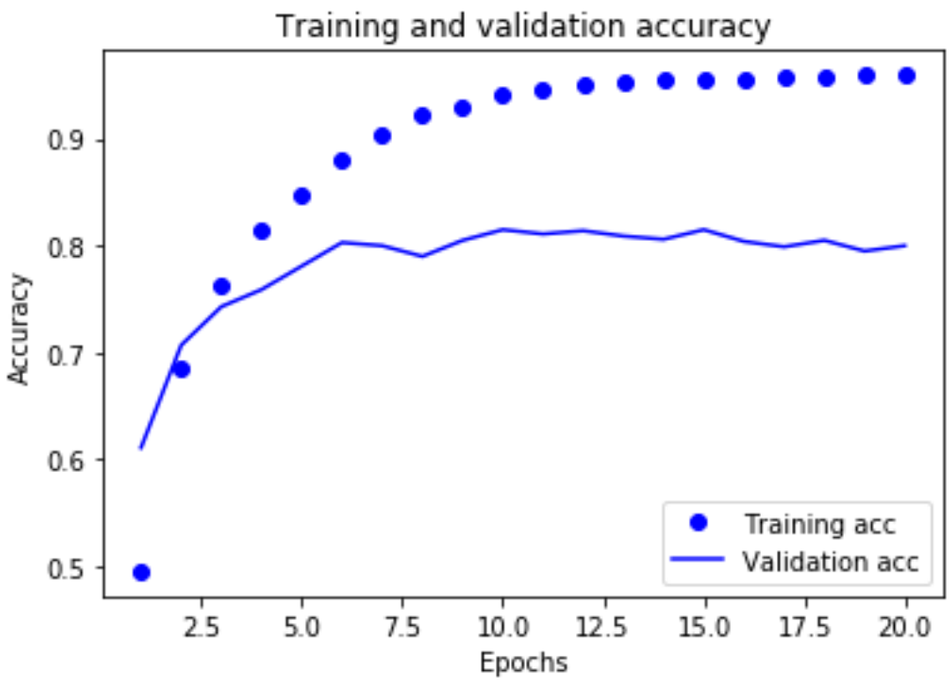


- 訓練データと検証データでの正解率をプロット

In [25]:
```
acc=history.history['acc']
val_acc=history.history['val_acc']

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

plt.plot(epochs,acc,'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()
```



8エポックの後、このネットワークは過学習に陥っています。

- モデルの訓練をやり直す

In [27]:
```
model=models.Sequential()
model.add(layers.Dense(64,activation='relu',input_shape=(10000,)))
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(46,activation='softmax'))

model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(partial_x_train,partial_y_train,epochs=8,batch_size=512,
          validation_data=(x_val,y_val))
results=model.evaluate(x_test,one_hot_test_labels)
```

```
Epoch 3/8
7982/7982 [==============================] - 4s 477us/step - loss: 1.0136 - acc: 0.77
81 - val_loss: 1.1303 - val_acc: 0.7530
Epoch 4/8
7982/7982 [==============================] - 4s 465us/step - loss: 0.7976 - acc: 0.82
51 - val_loss: 1.0539 - val_acc: 0.7590
Epoch 5/8
7982/7982 [==============================] - 4s 463us/step - loss: 0.6393 - acc: 0.86
24 - val_loss: 0.9754 - val_acc: 0.7920
Epoch 6/8
7982/7982 [==============================] - 4s 469us/step - loss: 0.5124 - acc: 0.89
21 - val_loss: 0.9102 - val_acc: 0.8140
Epoch 7/8
7982/7982 [==============================] - 4s 452us/step - loss: 0.4124 - acc: 0.91
37 - val_loss: 0.8932 - val_acc: 0.8210
Epoch 8/8
7982/7982 [==============================] - 4s 470us/step - loss: 0.3355 - acc: 0.92
90 - val_loss: 0.8731 - val_acc: 0.8260
2246/2246 [==============================] - 1s 486us/step
```

正解率は78%

In [28]:
```
results
```

Out[28]:
```
[0.9845026078653039, 0.784060552092609]
```

In [29]:
```
import copy
test_labels_copy=copy.copy(test_labels)
np.random.shuffle(test_labels_copy)
float(np.sum(np.array(test_labels)==np.array(test_labels_copy)))/len(test_labels)
```

Out[29]:
```
0.19679430097951914
```

3.5.5　新しいデータで予測値を生成する

- 新しいデータで予測値を生成する

```
In [31]: predictions=model.predict(x_test)
         predictions[0].shape
```

```
Out[31]: (46,)
```

各エントリは、長さが46のベクトルです。

```
In [32]: np.sum(predictions[0])
```

```
Out[32]: 1.0000001
```

ベクトルの係数を合計すると1です。

```
In [34]: np.argmax(predictions[0])
```

```
Out[34]: 3
```

確率が最も高いクラスです。

3.5.6　ラベルと損失値を処理する別の方法

In [35]:
```
y_train=np.array(train_labels)
y_test=np.array(test_labels)
model.compile(optimizer='rmsprop',loss='sparse_categorical_crossentropy',metrics=['acc'])
```

ラベルのリストを整数すのテンソルとしてキャストします。

数学的にはcategorical_crossentropyと同じであり、違いはインターフェイスだけです。

3.5.7　十分な大きさの中間層を持つことの重要性

- 情報ボトルネックを持つモデル

In [36]:
```
model=models.Sequential()
model.add(layers.Dense(64,activation='relu',input_shape=(10000,)))
model.add(layers.Dense(4,activation='relu'))
model.add(layers.Dense(46,activation='softmax'))

model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(partial_x_train,partial_y_train,epochs=20,batch_size=128,
          validation_data=(x_val,y_val))
```

```
Epoch 16/20
7982/7982 [==============================] - 4s 538us/step - loss: 0.7239 - acc: 0.78
69 - val_loss: 1.7184 - val_acc: 0.6600
Epoch 17/20
7982/7982 [==============================] - 4s 526us/step - loss: 0.7058 - acc: 0.79
58 - val_loss: 1.7715 - val_acc: 0.6610
Epoch 18/20
7982/7982 [==============================] - 4s 523us/step - loss: 0.6861 - acc: 0.80
18 - val_loss: 1.7718 - val_acc: 0.6690s: 0.6878 - acc: 0.801
Epoch 19/20
7982/7982 [==============================] - 4s 534us/step - loss: 0.6695 - acc: 0.80
87 - val_loss: 1.8187 - val_acc: 0.6750s: 0.6608  - ETA: 1s - loss: 0.6817 - acc: 0 -
ETA: 1s - los - ETA: 0s - loss: 0.6697 - acc: 0.808
Epoch 20/20
7982/7982 [==============================] - 4s 534us/step - loss: 0.6543 - acc: 0.81
40 - val_loss: 1.8318 - val_acc: 0.66603s - loss: 0.7053 - acc: 0 - ETA: 0s - loss:
0.6549
```

Out[36]: <keras.callbacks.History at 0x154009171d0>

検証データセットでの正解率のピークは大体71%であり、絶対損失は8%です。大量の情報を小さな中間層に詰め込みます。

3.5.8　そのほかの実習

- 32ユニットや128ユニットなど、さらに大きい中間層や小さい中間層を試してみる
- この例では、隠れ層を2つ使用している。隠れ層を１つにする、または３つにするとどうなるか試してみる。

3.5.9　まとめ

- データ点をN個のクラスに分類しようとしている場合、ネットワークの最後の層のサイスがNのDense層でなければならない。
- 多クラス単一ラベル分類問題では、ネットワークの最後の層では、活性化関数はSoftMaxを使用しべきである。
- 損失関数は多クラス交差エントロピーを使用しべきである。
- 多クラス分類でラベルを扱う方法は次の２つである。
  カテゴリエンコーディングー＞ categorical_crossentropy
  ラベルを整数としてエンコーディングー＞
  sparse_categorical_crossentropy
- 適当な中間層を使用することが重要である