計算機科学実験及演習4 画像処理 レポート1

橘大佑
1029-31-6811
2019年度入学

2021/11/30

工学部情報学科計算機科学コース
1029-31-6811
橘 大佑

計算機科学実験及び演習 4

レポート 1

2021 年 11 月 30 日

課題内容

MNIST の画像 1 枚を入力とし，3 層ニューラルネットワークを用いて，0〜9 の値のうち 1 つを出力するプログラムを作成した。

プログラムの説明

処理を、前処理・入力層・中間層への入力を計算する層 (全結合層)・出力層への入力を計算する層 (全結合層)・ソフトマックス関数・後処理の 6 つに分割して実装した。以下でそれぞれの役割について説明する。

- input_number 関数 (前処理)
  0 から 9999 までの整数を入力から受け取る。入力が整数でない場合や、整数であっても範囲外のものについては再び入力を促すようにした。また、実行する度に同じ結果を出力するよう乱数のシードを固定した。

Listing 1: input_data

```
def input_number():
  global number_of_image
  while(1):
    i = input('Input number(0~9999)!!!>>> ')
    if i.isdecimal():
     i = int(i)
     if 0 <= i and i < number_of_image:
      np.random.seed(seed=i)
      return i
     else:
      pass
```

- input_layer 関数 (入力層)
  MNIST のテストデータ 10000 枚の画像のうち、(input_numnber 関数で受け取った数字) 番目の画像を入力画像として用いた。この画像は 2 次元配列になっているため、それを 1 次元の配列に変換を行い、そこから 0 から 1 になるように正規化した。

Listing 2: input_layer

```
def input_layer(index):
  target_image = X[index]
  flattened_image = target_image.flatten()
  x = preprocessing.minmax_scale(flattened_image)
  return x
```

- fully_connected_layer_1 関数 (全結合層)
  入力層の線形和を入力として受け取り、次式で表される出力 $y_j^{(1)}$ を返す。

$$y_j^{(1)} = a\left(\sum_{i=1}^{j} w_{ji}^1 x_i + b_j^1\right)$$

ただし、関数 $a(t) = \frac{1}{1+\exp(-t)}$(シグモイド関数)

Listing 3: fully_connected_layer_1

```
def fully_connected_layer_1(input_vector):
  W_1 = np.random.normal(0, 1.0/len(input_vector), (M, len(input_vector)))
  b_1 = np.random.normal(0, 1.0/len(input_vector), (M, ))
  y_1 = expit(np.dot(W_1, input_vector) + b_1)
  return y_1
```

- softmax 関数 (ソフトマックス関数)
  C 個の入力を $a_i(i = 1, ..., C)$ とし、出力層の出力 $y_i^2(i = 1, ..., C)$ を次式で得る。

$$y_i^2 = \frac{\exp(a_i - \alpha)}{\sum_{j=1}^{C} \exp(a_i - \alpha)}$$

$$\alpha = \max(a_i)$$

  出力層の出力 $y_i^2(i = 1, ..., C)$ は入力 x がクラス i に属する尤度を表し、y_i^2 が最大となる i を認識結果 y として出力する。

Listing 4: softmax

```
def softmax(input_data):
  alpha = np.amax(input_data)
  return np.exp(input_data-alpha)/np.sum(np.exp(input_data-alpha))
```

- fully_connected_layer_2 関数 (全結合層)
  C 個のノードからなり、中間層の出力 y_j^1 の線形和を入力とする。

$$a_k = w_k^2 \cdot y^1 + b_k^2$$

Listing 5: fully_connected_layer_2

```
def fully_connected_layer_2(input_vector):
  W_2 = np.random.normal(0, 1.0/len(input_vector), (C, len(input_vector)))
  b_2 = np.random.normal(0, 1.0/len(input_vector), (C, ))
  y_2 = softmax(np.add(np.dot(W_2, input_vector), b_2))
  return y_2
```

- output 関数 (後処理)
  y_i^2 が最大となる i を認識結果 y として出力する.

Listing 6: output

```
def output_answer(input_data):
  print(np.argmax(input_data))
```

実行結果

以下が実行したときの出力である。

```
ebugpy\launcher" 55555 -- "c:\Users\Daisuke Tachibana\le4_python\kadai1.py" "
Input number(0~9999)!!!>>> str
Input number(0~9999)!!!>>> -56
Input number(0~9999)!!!>>> 999999
Input number(0~9999)!!!>>> 78
5
(base) C:\Users\Daisuke Tachibana\le4_pytho ...
Input number(0~9999)!!!>>> 78
5
(base) C:\Users\Daisuke Tachibana\le4_python ...
Input number(0~9999)!!!>>> 456
7
(base) C:\Users\Daisuke Tachibana\le4_python ...
Input number(0~9999)!!!>>> 456
7


(base) C:\Users\Daisuke Tachibana\le4_python> ...
```

”str”のような整数でない文字列や-56 や 999999 などの範囲外の整数が入力された場合は、再び入力するよう促されているのが分かる。また、同じ入力に関しては出力も同じになるようになっているのも分かる。78 という入力に関しては変わらず 5 と出力し、456 という入力に関しては常に 7 が出力されている。

工夫点

同じ入力に関しては出力も同じ (異なる出力をしない) になるように、乱数のシードを固定した。具体的には入力された数字をシードに固定した。

問題点

fully_connected_layer_1 関数と fully_connected_layer_2 関数は非常に似ているため、これらを一つに統合すればさらに見やすいコードになったと考えられる。