計算機科学実験及演習 4 画像認識
レポート 1

高橋駿一　2018 年度入学 1029-30-3949

提出日: 2020 年 10 月 15 日

課題 1

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.

作成したプログラムの説明

ソースコード 1: ライブラリのインポート

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import mnist
```

ライブラリのインポートを行った. なお, 本問では pyplot は活用しないが今後のことを考えてインポートした.

ソースコード 2: ライブラリのインポート

```
1  train_X = mnist.download_and_parse_mnist_file("train-images-idx3-ubyte.gz")
2  train_Y = mnist.download_and_parse_mnist_file("train-labels-idx1-ubyte.gz")
3  test_X = mnist.download_and_parse_mnist_file("t10k-images-idx3-ubyte.gz")
4  test_Y = mnist.download_and_parse_mnist_file("t10k-labels-idx1-ubyte.gz")
```

訓練データとテストデータの読み込みを行った.

ソースコード 3: 前処理

```
1  def preprocessing(N, M, d):
2      np.random.seed(seed=32)
3      W = np.random.normal(0, 1/N, (d, M))
4      b = np.random.normal(0, 1/N, (1, M))
5      return W, b
```

前処理を行うための関数である. N:手前の層のノード数 , M:中間層のノード数 , d:画像を表すベクトルの次元数として重み W と切片ベクトル b を分散 1/N, 平均 0 の正規分布に従う乱数で設定する. なお, 実行ごとに同じ結果を得るために seed 値は 32 で固定している.

ソースコード 4: 入力層の処理

```
1        def input_layer(X):
2            i = int(input())
3            input_image = np.array(X[i])
4            image_size = input_image.size
5            image_num = len(X)
6            class_num = 10
7            input_vector = input_image.reshape(1,image_size)
8            return input_vector, image_size, i, class_num
```

入力層の処理を行うための関数である．引数 X は MINIST の画像データであり, 標準入力で 0〜9999 の値を i として受け取り, X の i 番目のデータを入力画像とする．そしてこの画像データのサイズを image_size に格納し, 画像データを image_size 次元ベクトルの image_vector に変換し, これらの変数やクラス数 class_num を返す．

ソースコード 5: 線形和の計算

```
1        def matrix_operation(W, X, b):
2            return np.dot(X, W) + b
```

多次元の入力を受け取ってその線形和を出力する関数である．各ノードの入力 X, 重み W, 切片ベクトル b により計算を行う．

ソースコード 6: シグモイド関数

```
1        def sigmoid(x):
2            return (1 / (1 + np.exp(−1 ∗ x)))
```

引数 x にシグモイド関数を適用した値を返す関数である．多次元ベクトル x をそのまま扱うことができる．

ソースコード 7: ソフトマックス関数

```
1        def softmax(a):
2            alpha = np.amax(a)
3            exp_a = np.exp(a − alpha)
4            sum_exp = np.sum(exp_a)
5            y = exp_a / sum_exp
6            return y
```

引数 a にソフトマックス関数を適用した値を返す関数である．

ソースコード 8: 後処理

```
1        def postprocessing(y):
2            binary_y = np.where(y == np.amax(y), 1, 0)
3            print(np.where(binary_y == 1)[1][0])
4            return binary_y
```

後処理を行うための関数である. 出力層の値をベクトルとして受け取り, 値が一番大きいものを 1, それ以外の値を 0 に変換し, 1 に変換された値に対応するインデックス (0〜9) を標準出力に出力する.

ソースコード 9: 課題 1 の実行

```
1        input_vec, image_size, i, class_sum = input_layer(test_X)
2        # print('input', image_size, i, class_sum )
3        W1, b1 = preprocessing(image_size, 30, image_size)
4        y1 = matrix_operation(W1, input_vec, b1)
5        # print('matrix', y1)
6        y1 = sigmoid(y1)
7        # print('sigmoid', y1)
8        W2, b2 = preprocessing(30, class_sum, 30)
9        a = matrix_operation(W2, y1, b2)
10       # print('a', a)
11       y2 = softmax(a)
12       # print(y2)
13       binary_y = postprocessing(y2)
14       # print(binary_y)
```

ここまでに作成した関数を活用して課題 1 の処理を実行した. なお, 中間層のノード数は 30 とした.

実行結果

標準入力に 400 を入力すると 2 が標準出力に出力された. コメントアウトを外すことで各段階でも正しく動作していることが確認できた.

工夫点

- この後の課題でも活用しやすくするために前処理, 入力層の処理, 線形和の計算, シグモイド関数, ソフトマックス関数, 後処理を別々の関数として実装した.
- 入力層の処理を行う input_layer(X) で画像のサイズやクラス数を後の実装のために変更が可能な仕様にした.
- 線形和の計算を行う matrix_operation は中間層への入力を計算する層と出力層への入力を計算する層のそれぞれの計算に対応している.
- ソフトマックス関数を適用する softmax で for 文を使わず多次元ベクトルのまま処理できるように記述した.

問題点

- postprocessing(y) でベクトルのインデックスを標準出力に出力しているが, このままでは分類するクラスが 0 からではない場合は正しくない値を出力してしまう.
- この後の課題で実装することではあるが重み W を乱数で設定しているのでこのニューラルネットワークでは正しい結果は得られない.