計算機科学実験4 画像処理
課題1 レポート

1029311501 坂井優斗

提出日：2021年12月9日

1　課題内容

MNIST の画像 1 枚を入力として，3 層のニューラルネットワークを用いて，0 ～ 9 から値を 1 つ出力するプログラムを作成する。プログラムの仕様は以下の通りである。

- 入力として, 0 ～ 9999 の整数 i を 1 つ受け取り, ニューラルネットワークに通したのち 0 ～ 9 の整数 1 つを出力する。
- 使用する MNIST の画像は 10000 枚のテストデータのうち, i 番目を使用する。
- 中間層のノードの数 M は自由 → 100 とした。
- 重み $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ は, 手前の層のノードの数 N の逆数 $1/N$ を分散とする正規分布によって定められる乱数で決定する。

2　プログラムの説明

2.1　グローバル変数

- input_node_size = 784 : 入力層のノードの数
- inner_node_size = 5 : 中間層のノードの数
- output_node_size = 10 : 出力層のノードの数

2.2　main 関数

Code 1: main 関数

```
1   def main():
2     test_images = mnist.download_and_parse_mnist_file("t10k-images-idx3-ubyte.gz")
3     test_labels = mnist.download_and_parse_mnist_file("t10k-labels-idx1-ubyte.gz")
4
5     i = int(input())
6     image = test_images[i]
7     processed_image = output_layer(inner_layer(input_layer(image)))
8     prediction = np.argmax(processed_image)
9     print(prediction)
```

まず, 2, 3 行目で test_images, test_labels という名前の変数に MNIST のテスト画像データと正解ラベルを読み込み, 保存する。
次に, 5 行目で input() 関数を用いて, 標準入力で整数を受け取り, それを i とする。6 行目でテスト画像が格納されている配列 test_images から i 番目の画像をとってきて, 7 行目でその画像を 3 層からなるニューラルネットワークに入力する。
入力層, 中間層, 出力層はそれぞれ input_layer, inner_layer, output_layer 関数で定義されているので, これらの関数に入力することでニューラルネットワークによる処理が行える。処理を行なったデータを processed_image として, この配列の要素は各ラベルである尤度を示しているので, numpy の argmax 関数で尤度が最大であるインデックスを取得し, その値を標準出力に出力している。

2.3 sigmoid_function

Code 2: sigmoid_function

```
1  def sigmoid_function(t):
2    return 1/(1 + np.exp(-t))
```

この関数は,

$$f(\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{x})}$$

で表されるシグモイド関数を実装している。

2.4 softmax_funciton

Code 3: softmax_function

```
1  def softmax_function(a):
2    max_a = a.max()
3    sum = np.sum(np.exp(a - max_a))
4    return np.exp(a - max_a) / sum
```

この関数は, 入力を $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, 出力を $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ とすると,

$$y_i = \frac{\exp(x_i - \alpha)}{\sum_{j=1}^{n} \exp(x_j - \alpha)}$$

$$\alpha = \max x_i$$

上の式で表されるソフトマックス関数を実装している。通常のソフトマックス関数とは違い, max 関数を利用して, 入力ベクトルの全要素のうち最大値 α をとり, 各要素から α を引いてから冪乗をとっている。これは, 冪乗をとったときのオーバーフローの対策をするためである。

2.5 input_layer

Code 4: input_layer

```
1  def input_layer(image):
2    image = np.array(image)
3    trans_image = np.reshape(image, (input_node_size, 1))
4    return trans_image
```

これは 3 層ニューラルネットワークにおける入力層の働きを持つ関数である。入力された画像を reshape 関数を利用して, input_node_size 次元のベクトルに変換して出力する。今回の実験では, MNIST のデータを用いていて, これらの画像は全て 28×28 の 2 次元配列で表されているので, inner_layer では input_node_size $= 28 \times 28 = 784$ としている。

2.6 inner_layer

Code 5: inner_layer

```
def inner_layer(x):
  np.random.seed(10)
  W_1 = np.random.normal(loc=0, scale=np.sqrt(1/input_node_size), size=(inner_node_size
    , input_node_size))
  b_1 = np.random.normal(loc=0, scale=np.sqrt(1/input_node_size), size=(inner_node_size
    , 1))
  return sigmoid_function(np.dot(W_1, x) + b_1)
```

これは 3 層ニューラルネットワークにおける中間層の働きを持つ関数である。はじめに, 乱数に再現性を持たせるために np.random.seed(10) でシード値を 10 に固定している。その後, 中間層での重み $W^{(1)}$ とバイアス項 $\boldsymbol{b}^{(1)}$ を乱数で生成している。numpy.random.normal() 関数を利用することで正規分布に従う乱数から行列を生成することができる。第一引数の”loc=”では平均が, 第二引数の”scale=”では標準偏差が指定できるので, 今回は課題の仕様通り平均を 0, 標準偏差を一つ前の層のノード数の逆数の平方根, すなわち $\sqrt{\frac{1}{input\_node\_size}}$ に設定している。
そして, 入力 $\boldsymbol{x}$ に対して, numpy の dot 関数を利用して, 線形和 $W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$ を計算する。最後に, 活性化関数として 2.3 で定義したシグモイド関数を用いて, $W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$ を入力としたシグモイド関数の出力を, inner_layer 関数の返り値としている。

2.7 output_layer

Code 6: output_layer

```
def output_layer(y):
  np.random.seed(20)
  W_2 = np.random.normal(loc=0, scale=np.sqrt(1/inner_node_size), size=(
    output_node_size, inner_node_size))
  b_2 = np.random.normal(loc=0, scale=np.sqrt(1/inner_node_size), size=(
    output_node_size, 1))
  return softmax_function(np.dot(W_2, y) + b_2)
```

これは 3 層ニューラルネットワークにおける出力層の働きを持つ関数である。中間層の機能をもつ inner_layer 関数と同じように, 乱数の生成に再現性を持たせるために np.random.seed() 関数でシード値を 20 に固定している。その後, 出力層での重み $W^{(2)}$ とバイアス項 $\boldsymbol{b}^{(2)}$ を乱数で生成している。これも中間層の時と同じように numpy.random.normal() 関数を利用して, 正規分布に従う乱数から行列を生成している。平均を 0, 標準偏差を一つ前の層のノード数の逆数の平方根, すなわち $\sqrt{\frac{1}{inner\_node\_size}}$ に設定している。
そして, 入力 $\boldsymbol{y}$ に対して, 線形和 $W^{(2)}\boldsymbol{y}+\boldsymbol{b}^{(2)}$ を計算する。最後に, 活性化関数として 2.4 で定義したソフトマックス関数を用いて, $W^{(2)}\boldsymbol{y}+\boldsymbol{b}^{(2)}$ を入力としたソフトマックス関数の出力を, output_layer 関数の返り値としている。

3 実行結果

コマンド `python task1.py` でスクリプトを実行する。
テスト用画像の何番目の画像に対してクラス識別を行うかを標準入力から受けとるので, 整数を入力する。ここでは例として”0”を入力する。すると, どのクラスに識別されたかが標準出力で表示される。以下のように実行される。

ターミナル

```
$ python task1.py
0
7
```

4 工夫点

- numpy が提供している行列演算の関数を利用することで, for 文を回すことなく行列の内積等の計算を行った。これによって計算時間の短縮が可能となった。
- ニューラルネットワークにおける機能を関数ごとに切り分けることで, 可読性を高めた。また, これによってプログラムの変更や拡張が容易に行えるようになっている。

5 問題点

- ニューラルネットワークのパラメータがただの乱数であるので, クラス識別の精度が非常に悪い。これはパラメータの学習を行うことで解決できる問題である。