画像実験課題 1

1029323422 天野岳洋

2022 年 12 月 10 日

1　概要

課題 1 の内容は 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せよ．というものである

2　要件定義

- キーボードから 0 9999 の整数 i を入力として受け取り,mnist のテストデータの i 番目を三層ニューラルネットワークの入力とする.
- 三層ニューラルネットワークの構造は以下のとおりである. なお中間層, 最終層のノードの数を M とする
  1. 前処理: 画像の shape を (28, 28) から (784,) に変更する
  2. 一層目: 入力をそのまま出力とする
  3. 全結合層 1: shape が (M, 784) である重み W1 と (M,1) であるバイアス b1 を用いる
  4. 中間層: 活性化関数としてシグモイド関数を用いる
  5. 全結合層 2: shape が (C, M), (C, 1) であること以外は 3. と同じ
  6. 出力層: 活性化関数としてソフトマックス関数を用いる
  7. 後処理: 出力層の出力の中で最大値をとるものの位置を最終出力とする
- 三層ニューラルネットワークの最終出力である 0〜9 の整数を print する
- 上の中で用いた変数 W1, b1, W2, b2 は適切な方法でランダムにとる. なおこのランダムシードはある値で固定する

3　実装

このセクションではプログラムの作成方針と実際のコードを順に説明する。

3.1　準備

Listing 1　preparation

```
1  import numpy as np
2  import mnist
3
4  M = 30
5  C = 10
6  seed = 10
7  np.random.seed(seed)
8  X = mnist.download_and_parse_mnist_file("t10k-images-idx3-ubyte.gz")
```

ここでは必要なパッケージの import と中間層, 最終層, randomseed の設定を行っている. なお中間層の数は適当に取ったものであり, 最終層の数は分類したいクラス (0〜9) の数に等しい. そして,X として (?, 28, 28) の MNIST テストデータの画像配列を取得している.

3.2 標準入力

Listing 2 stdin

```
print("0~" + str(len(X)-1))
idx = int(input())
before_conv = np.array(X[idx])
```

一行目では入力としてエラーにならないもの (list の長さより小さいもの) の範囲を表示している. その後、標準入力を受け取り整数化を行い, X の idx 番目を取得する. ここで標準入力として X の長さ以上のものを受け取るとエラーだが、その際の処理は考えていない.

3.3 前処理

ここでは重みバイアスの初期化, 画像の shape の変更を行っている。

Listing 3 pre-processing

```
img_size = before_conv.size
img = before_conv.reshape(img_size, 1)

W1 = np.random.normal(loc = 0, scale = np.sqrt(1/img_size), size=(M,
    img_size))
b1 = np.random.normal(loc = 0, scale = np.sqrt(1/img_size), size=(M, 1))
W2 = np.random.normal(loc = 0, scale = np.sqrt(1/M), size=(C, M))
b2 = np.random.normal(loc = 0, scale = np.sqrt(1/M), size=(C, 1))
```

まず img_size を取得する (=784). その後 before_conv の shape を (img_size, 1) に変換したものを img として三層ニューラルネットワークに与える. ここで img は (0, 0, 0, ...).T のような形になっている。そして平均が 0 で分散が 1/前層ノードの数となるようにランダムに重みバイアスを作成する. np.random.normal の scale には標準偏差を入れるので $\sqrt{}$ 処理を行っている。

3.4 三層ニューラルネットワーク

Listing 4 3NN

```
input1 = np.dot(W1, img) + b1
output1 = 1/(1 + np.exp(-1 * input1))
input2 = np.dot(W2, output1) + b2
```

```
4   alpha = input2.max()
5   sumexp = np.sum(np.exp(input2 - alpha))
6   output_last = np.exp(input2 - alpha) / sumexp
```

全結合層 1 では入力を X として $Y = WX + b$ を出力している.Y の shape が (M, 1) となるように W,b の shape を定める.
中間層では先ほどの Y の全要素それぞれを入力 x として $Y = \frac{1}{1+e^{-x}}$ を出力している. numpy ではこの全要素に対して同一の関数を適用させるという動作を自動で実行するのでそのまま行列 X を入力としてよい.
全結合層 2 は先ほどと同様である。出力層ではソフトマックスの alpha を取得し, ソフトマックス関数の分母を計算してから, それぞれの要素に対してソフトマックス関数を適用させている.

3.5 後処理

Listing 5 after-processing

```
1   answer = np.argmax(output_last)
2   print(answer)
```

output_last は次のようなものである.

$$output\_last = [0.1, 0.05, 0.2, ...].T$$

この行列の意味は 0 番目の 0.1 は入力の数字が 0 である確率が 0.1 であることを意味しており, この行列の最大値をとる位置が 3NN によって推論された数字である. なので np.argmax の出力が推論された数字そのものである. 最後にその数字を print している.

4 実際の動作

Listing 6 result

```
1   0~9999
2   0
3   9
```

0〜9999 が表示され、0 を標準入力として与えたものである。そして 9 が標準出力として得られた。このように期待していた通りの結果となった。