計算機科学実験４レポート 必修課題１

工学部情報学科３回生 計算機科学コース
1029310587 谷綜一郎

2021 年 12 月 13 日

目 次

第1章　課題内容

1.1　概要

「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。」

1.2　諸条件

- キーボードから 0〜9999 の整数を入力 i として受け取り、0〜9 の整数を標準出力に出力すること。
- MNIST のテストデータ 10000 枚の画像のうち i 番目の画像を入力画像として用いる。
- MNIST の画像サイズ（28 × 28）、画像枚数（10000 枚）、クラス数（C = 10）は既知とする。ただし、後々の改良のため変更可能な仕様にしておくことを薦める。
- 中間層のノード数 M は自由に決めて良い。
- 重み $W^{(1)}$, $W^{(2)}$, $\boldsymbol{b}^{(1)}$, $\boldsymbol{b}^{(2)}$ については乱数で決定すること。ここでは、手前の層のノード数を N として 1/N を分散とする平均 0 の正規分布で与えることとする。適切な重みを設定しないため、課題 1 の段階では入力に対してデタラメな認識結果を返す。ただし、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すること。

1.3　実装の方針

後の課題のために、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）・シグモイド関数・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、に分割して実装する。

- 「入力層」では MNIST の画像を 784 次元のベクトルに変換する。
- 「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」は、パラメータは違えど処理は同じ（全結合層）。多次元の入力を受け取ってその線形和を出力する。
- 「シグモイド関数」の実装にあたっては、多次元ベクトルを入力とできるようにする。python では for 文を用いると極端に処理速度が遅くなるので、for 文を使わずに済む工夫をする。

第2章　プログラム説明

2.1　概要

先述した課題内容の通り、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 **program1.py** を作成した。なお実際には、0〜9999 までのある数値を標準入力として受け取り、画像 10000 枚のテストデータの中から、その数値に対応した画像を入力データとしている。

まず、実装の基礎となる 3 層ニューラルネットワークについて簡単に説明した後、プログラムを動作させる際の外部仕様、実装したコードの内部設計について記述する。

2.2　３層ニューラルネットワーク

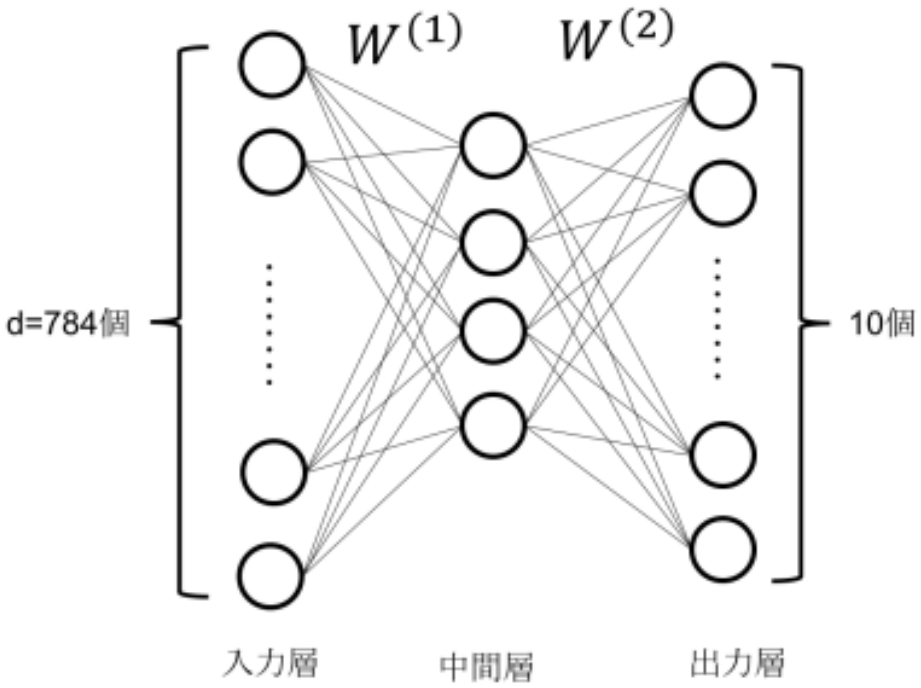

図 2.1: 3 層ニューラルネットワークの模式図

模式図を図 2.1 に示した通り、3 層ニューラルネットワークは、以下の 3 層で構成される。

- 入力次元数 d と同じ数のノードからなる入力層
- クラス数（今回は 0〜9 までの 10 クラス）と同じ数のノードからなる出力層
- 入力層と出力層の間にある中間層

入力層の各ノードは入力 $\boldsymbol{x}$ の各要素 $x_i(i=1,...,d)$ に対応し、x_i を入力として x_i をそのまま出力する。中間層は M 個のノードからなり、中間層の各ノード $j(j=1,...,M)$ は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。

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

ここで、関数 $a(t)$ は**活性化関数**と呼ばれ、**シグモイド関数**

$$a(t) = \frac{1}{1+exp(-t)}$$

が良く用いられている。線形和の重みを $\boldsymbol{w_j^{(1)}}=(w_{1j}^{(1)},...,w_{dj}^{(1)})^T$ とすれば、式 1 は $\boldsymbol{w_j^{(1)}}$ と $\boldsymbol{x}$ との内積を用いて以下のようにシンプルに書ける。

$$y_j^{(1)} = a(\boldsymbol{w}_j^{(1)} \cdot \boldsymbol{x} + b_j^{(1)})$$

さらに，線形和の重み $\boldsymbol{w_j^{(1)}}$ を、j 行目の成分とする M 行 d 列の行列 $W^{(1)}$ と M 次元ベクトル $\boldsymbol{b^{(1)}}=(b_1^{(1)},...,b_d^{(1)})^T$ を用いて中間層への M 個の入力を $W^{(1)}\boldsymbol{x} + \boldsymbol{b^{(1)}}$ と書くことができる。

出力層は C 個（C はクラス数）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。

$$a_k = \boldsymbol{w_k^{(2)}} \cdot \boldsymbol{y^{(1)}} + b_k^{(2)}$$

ここで、$\boldsymbol{y^{(1)}} =(y_1^{(1)},...,y_M^{(1)})^T$ である。また、C 個の入力 $\boldsymbol{a} =(a_1^{(1)},...,a_C^{(1)})^T$ を、線形和の重み $\boldsymbol{w_k^{(2)}}$ を k 行目の成分とする C 行 M 列の行列 $W^{(2)}$ と C 次元ベクトル $\boldsymbol{b^{(2)}} =(b_1^{(2)},...,b_d^{(2)})^T$ を用いて、$\boldsymbol{a} = W^{(2)}\boldsymbol{y}^{(1)} + \boldsymbol{b}^{(2)}$ と書くことができる。

出力層における活性化関数として**ソフトマックス関数**が用いられる。C 個の入力を $a_i\ (i=1,...,C)$ とし、ソフトマックス関数を用いて出力層の出力 $y_i^{(2)}\ (i=1,...,C)$ を次式で得る。ここで、ソフトマックス関数の定義では α は存在しないが、数値計算時のオーバーフローに対処するため α を導入している。

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

$$\alpha = maxa_i$$

出力層の出力 $y_i^{(2)}\ (i=1,...,C)$ は入力 $\boldsymbol{x}$ 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。

2.3 外部仕様

0〜9999 までの整数値 i を入力することで、i 番目の画像テストデータに対して 0〜9 のうちいずれかの数値を出力する。

まず、「Please enter the number as you like. (0 ～ 9999)」と表示されるため、 0〜9999 までの整数値 i を入力する。この時、文字や 0〜9999 に含まれない値など、不適切なものが入力された場合は、エラーメッセージとともにもう一度入力を促される。

適切な整数値が入力されると、i 番目の画像テストデータが別ウィンドウで表示される。また、「result: 」の後ろに、0〜9 のうち 1 つの数値が出力され、「answer: 」の後ろに、i 番目の画像に対する実際の正解ラベルである、0〜9 のうち 1 つの数値が出力される。

2.4 内部設計

2.4.1 概要

1.3 実装の方針 でも記述した通り、「前処理・入力層・中間層への入力を計算する層（全結合層）・シグモイド関数・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」に分割して、それに対応する諸関数を定義することで、プログラムを実装した。グローバル変数について説明した後に、各ステップに分けて、諸関数に関する内部設計を記述する。また、最終的に実行する関数 $main1()$ を、以下のプログラム 2.1 に示す。

プログラム 2.1: main1()

```
1  def main1():
2    (img, ans) = preprocess()
3    (w1, b1, w2, b2) = preWB()
4    x = inputLayer(img)
5    y1 = sigmoid(combine(x, w1, b1))
6    a = combine(y1, w2, b2)
7    y2 = softmax(a)
8    max = np.argmax(y2)
9    print("result: " + str(max))
10   print("answer: " + str(ans))
11   plt.imshow(img, cmap=cm.gray)
12   plt.show()
```

このコード中の変数については以下のとおりである。

- img：28 × 28 の画像データ（2 次元配列）
- ans：正解ラベルデータ（0〜9 の整数値）
- w1: 100 × 784 の二次元配列
- b1: 100 × 1 の二次元配列
- w2: 10 × 100 の二次元配列
- b2: 10 × 1 の二次元配列
- x：784 × 1 の画像データ
- y1：100 × 1 の二次元配列
- a：10 × 1 の二次元配列
- y2：10 × 1 の二次元配列
- max：y2 のうち最大の数値のインデックス（0〜9 の整数値）

2.4.2 グローバル変数

- X : 10000 × 28 × 28 の画像データ（3 次元配列）
- Y : 10000 個の正解ラベルデータリスト（1 次元配列）
- ImageSize : MNIST の画像サイズに関する定数（ここでは 28）
- D : ImageSize の 2 乗で定義した定数（ここでは 784）
- ImageNum : 画像テストデータの枚数（ここでは 10000）
- ClassNum : 出力層におけるノード数（ここでは 10）
- MiddleNodeNum : 中間層におけるノード数（ここでは 100）

2.4.3 前処理

- preprocess, imputNumber

具体的なコードは、以下のプログラム 2.2, 2.3 に示すとおりである。

preprocess は、inputNumber を用いて、標準入力から 0〜9999 の整数値 i を受けとり、X における i 番目の画像テストデータ $image28$（28 × 28 の二次元配列）と Y における i 番目の正解ラベル ans(0〜9) を返す。ここで、入力された i を用いて、乱数シードを設定する。

プログラム 2.2: preprocess

```
1  def preprocess():
2   i = inputNumber()
3   np.random.seed(i)
4   image28 = X[i]
5   ans = Y[i]
6   return image28, ans
```

プログラム 2.3: inputNumber

```
1  def inputNumber():
2   num = input("Please enter the number as you like. (0 ~ " + str(
        ImageNum - 1) + ")\n")
3   try:
4       num = int(num)
5   except:
6       print("Please enter the 'number'.")
7       return inputNumber()
8   if num < 0 or num > (ImageNum - 1):
9       print("Please enter '0 ~ " + str(ImageNum - 1) + "'.")
10      return inputNumber()
11  else:
12      return num
```

- preWB(), random(row, column, preNodeNum)

  具体的なコードは、以下のプログラム 2.4, 2.5 に示すとおりである。

  preWB は、preprocess で設定した乱数シードと random を用いて、以下の 4 つの二次元配列を返す。

  - w1: 100 × 784 の二次元配列
  - b1: 100 × 1 の二次元配列
  - w2: 10 × 100 の二次元配列
  - b2: 10 × 1 の二次元配列

  ここで、random は、3 つの数値 $row, column, preNodeNum$ を入力として、$1/preNodeNum$ を分散とする平均 0 の正規分布で与えられた $row \times column$ 個の乱数を $row \times column$ の二次元配列の形にした $array$ を返す。

プログラム 2.4: preWB

```
1  def preWB():
2   w1 = random(MiddleNodeNum, D, D)
3   b1 = random(MiddleNodeNum, 1, D)
4   w2 = random(ClassNum, MiddleNodeNum, MiddleNodeNum)
5   b2 = random(ClassNum, 1, MiddleNodeNum)
6   wb = w1, b1, w2, b2
7   return wb
```

プログラム 2.5: random

```
1  def random(row, column, preNodeNum):
2   array = np.random.normal(
3               loc = 0,
4               scale = math.sqrt(1/preNodeNum),
5               size = (row, column),
6           )
7   return array
```

2.4.4 入力層

- inputLayer

  具体的なコードは、以下のプログラム 2.6 に示すとおりである。

  inputLayer は、28 × 28 の二次元配列 $image$ を入力として、配列しなおすことで、784 × 1 の二次元配列 $image784$ を返す。

プログラム 2.6: inputLayer

```
1   def inputLayer(image):
2    image784 = image.reshape(D, 1)
3    return image784
```

2.4.5 結合層

- combine

具体的なコードは、以下のプログラム 2.7 に示すとおりである。

combine は、$q \times r$ の二次元配列 $input$ と $p \times q$ の二次元配列 $weight$ と $p \times r$ の二次元配列 b を入力として、$weight$ と $input$ の行列積に b を加えたものを計算して、$p \times r$ の二次元配列を返す。

プログラム 2.7: combine

```
1   def combine(input, weight, b):
2    return np.dot(weight, input) + b
```

2.4.6 シグモイド関数

- sigmoid

具体的なコードは、以下のプログラム 2.8 に示すとおりである。

多次元配列 t を入力として、各要素に**シグモイド関数**を適用したような、t と同じ形の配列を返す。

プログラム 2.8: sigmoid

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

2.4.7 ソフトマックス関数

- softmax

具体的なコードは、以下のプログラム 2.9 に示すとおりである。

多次元配列 x を入力として、**ソフトマックス関数**を適用した後に、x と同じ形の配列を返す。

プログラム 2.9: softmax

```
1  def softmax(x):
2   y = np.exp(x - np.max(x))
3   f_x = y / np.sum(y)
4   return f_x
```

2.4.8 後処理

- 標準出力

具体的なコードは、以下のプログラム 2.10 に示すとおりである。なお、このコードはプログラム 2.1 の一部である。

max は、10 × 1 の二次元配列である $y2$ のうち最大の数値のインデックス（0〜9 の整数値）であり、3 層ニューラルネットワークによって得られた計算結果を表す。「result:」の後ろに出力する数値を、「answer:」の後ろに選ばれた画像に対する実際の正解ラベルの数値を出力する。どちらも、0〜9 の整数値である。また、選ばれた画像データも別ウィンドウに表示する。

プログラム 2.10: 標準出力

```
1  ...
2   max = np.argmax(y2)
3   print("result: " + str(max))
4   print("answer: " + str(ans))
5   plt.imshow(img, cmap=cm.gray)
6   plt.show()
7  ...
```

第3章　実行結果

標準入力として 4000 を入力した場合の実行結果をプログラム 3.1 に示す。また、実行の際に表示されている画面を図 3.1 に示す。入力が不適切だった場合のエラーもうまく動いていることが分かる。

プログラム 3.1: 実行結果

```
>python program1.py
Please enter the number as you like. (0 ~ 9999)
a
Please enter the 'number'.
Please enter the number as you like. (0 ~ 9999)
40000
Please enter '0 ~ 9999'.
Please enter the number as you like. (0 ~ 9999)
4000
result: 0
answer: 9
```

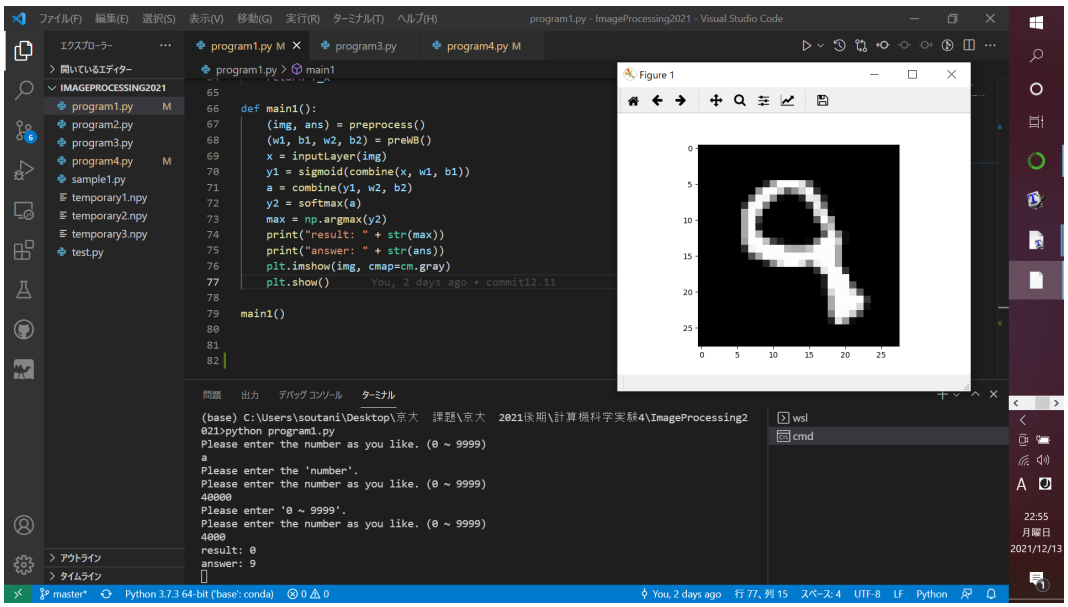

図 3.1: 実行結果

第4章　工夫点

前処理

数値入力の際のエラー対応などの機能を追加した。

また、実行する度に同じ結果が出力されるように乱数シードを設定する際に、標準入力で得られる i を用いた。

後処理

画像テストデータに対して、0〜9 のうち 1 つの数値を出力するだけでなく、比較として、正解ラベルの数値と選ばれた画像データも出力するようにした。

全体

3 層ニューラルネットワークのステップや関数ごとに、細かく分割してプログラムを実装して、見た目などの可読性も考慮して記述した。

第5章　問題点

全体

今回作成した一連のプログラムを、複数回続けて行えるような拡張が考えられる。画像が表示された別ウィンドウを閉じた後に、もう一度数値入力あるいはプログラムを続けるかどうかを促すような機能があれば、デバッグや機能拡張の際に有用である。