計算機科学実験及演習 4
画像認識 第 1 回レポート

谷　勇輝

入学年 平成 27 年
学籍番号 1029-27-2870

締切日: 2017 年 12 月 22 日
提出日: 2017 年 12 月 21 日

[課題 1] 3 層ニューラルネットワークの構築（順伝播）

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．

- 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), b(1), 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 作成したプログラム

1.1 プログラム構成

今回作成したプログラムは以下のモジュール、クラスからなる。

オブジェクト指向・ドメインモデルに基づいた適切なモジュール設計により、今後の開発がよりスムーズになるよう注意した。

```
neural_network
 ├ ioex.py          … 入出力系モジュール
 |  ├ InputManager      … 入力を担当するクラス
 |  └ OutputManager     … 出力を担当するクラス
 ├ data.py          … プログラムで扱う基本データ系モジュール
 |  ├ MnistDataBox      … MNIST データ群を表すクラス
 |  └ MnistData         … 単一の MNIST データを表すクラス
 ├ layer.py         … ニューラルネットの層モジュール
 |  ├ Layer             … 基本の層構造を表すベースクラス
 |  ├ InputLayer        … 入力層を表すクラス
```

```
┃  ┣ ConversionLayer     … 変換を行う層を表すクラス
┃  ┣ HiddenLayer         … 中間層を表すクラス
┃  ┗ OutputLayer         … 出力層を表すクラス
┣ util.py                … 汎用関数や汎用クラスを集めたモジュール
┃  ┣ ComplexMaker        … numpy の random を扱うクラス
┃  ┗ 各種数学メソッド
┣ test.py                … 動作確認・テスト用モジュール
┗ task1.py               … 課題 1 プログラム
MNIST
```

1.2 メインプログラム

課題 1 のメインプログラム test1.py は以下の通りである。入力は MNIST データの番号、出力は学習前のランダムな重みによる 3 層ニューラルネットワークの出力である。追加の機能として、入出力を任意回数行えるように変更した。

test1.py

```
1    import ioex
2    import layer
3    import util
4
5    print("### task1 ###")
6
7    # 入出力準備
8    inputM = ioex.InputManager()
9    outputM = ioex.OutputManager()
10   testingData = inputM.getMnistsTestingData()
11
12   # 3層ニューラルネットワーク
13   inputLayer = layer.InputLayer(28*28)
14   hiddenLayer = layer.HiddenLayer(50,inputLayer,1,1)
15   outputLayer = layer.OutputLayer(10,hiddenLayer,1,1)
16
17   #入出力
18   loop = True
19   while(loop):
20       # 入力
21       targetNum = inputM.selectNumber()
22       sample = testingData.getSingleData(targetNum)
23       inputLayer.setInput(sample.getImage())
24
25       # 出力
26       result = outputLayer.calculate()
27       outputM.printMaxLikelihood(result)
28       print(result)
29
30       print("\ncontinue?")
31       loop = inputM.selectYesOrNo()
32
33   print("Bye.")
```

13 行目から 15 行目では、3 層ニューラルネットワークの作成を行っている。各 Layer クラスにおいて、第 1 引数はその層における次元の数 (ニューロンの数)、第 2 引数は前の層のインスタンス、第 3 引数はニューロンの入力にかかる重みの初期値を決定する乱数シード、第 4 引数はニューロンの入力

に加算される閾値の初期値を決定する乱数シードである。各 Layer インスタンスは生成された時にコンストラクタによって初期化され、その際に各重みの値は指定した乱数シードに基づきランダムに決定される。

入力後のプログラムの流れを示す。まず、23 行目の InputLayer#setInput メソッド によってニューラルネットに入力がセットされる。次に、26 行目の OutputLayer#calculate メソッド によってニューラルネットワーク内の値が順方向に更新される。最後に、27 行目の OutputManager#printMaxLikelihood メソッドによって 出力層の結果の中で最大の値をもつニューロンを特定しその番号を出力している。

1.3 layer モジュール

ニューラルネットワークの部品となる layer モジュール内のクラスはこのプログラム群において重要な役割を果たしている。その詳細を以下に示す。

1.3.1 Layer クラス

このモジュール内の全ての Layer クラスのベースクラスであり、他の Layer はこのクラスを必ず継承する。層についての基本の機能を備えている。

主なインスタンス変数は以下の通りである。

- self.dimension

  層に含まれるニューロンの数、すなわち層の次元数を表す。コンストラクタの引数によって初期化される。

- self.output

  層の出力を表す。caluculate メソッドによって更新され、現在の最新の出力情報を保持する。初期値は全て 0 である。

また、主なメソッドは以下の通りである。

- __init__(self, dimension)

  インスタンスを作成し値を初期化する。層に含まれるニューロンの数 (次元数) はここで決定する。

- calculate(self)

  現在の最新の情報を使用して出力を更新する。更新後の値を返す。

- getOutput(self)

  現在の最新の出力情報を返す。このメソッドでは出力値の更新は行われない。

- confirmParameters(self)

  層についての情報を標準出力に表示する。

- getDimension(self)

  層が含むニューロンの数、すなわち次元数を返す。

1.3.2 InputLayer

入力層を表すクラス。Layer クラスを継承する。
重要なインスタンス変数は以下の通りである。

- self.input

  入力を表す変数。初期値は全て 0 である。setInput メソッドによって設定する。

重要なメソッドは以下の通りである。

- setInput(self, inputData)

  入力を設定する。引数に与えられた配列は 1 次元に圧縮される。

- calculate(self)

  Layer クラスのメソッドをオーバーライドしている。現在の入力の値をそのまま出力として設定する。

1.3.3 ConversionLayer

入力に対し重みをつけ、さらに何らかの活性関数を適応したものを出力とする層、すなわち何らかの変換を行う層を表す。具体的な実装としては後に示す HiddenLayer や OutputLayer がある。
重要なインスタンス変数は以下の通りである。

- self.prevLayer

  直前の層のインスタンスを保持する。コンストラクタの引数によって初期化される。

- self.weightSize

  入力に掛けられる重み行列のサイズを表したタプル。前の層の次元情報とこの層の次元情報から自動的に初期化される。

- self.shiftSize

  入力に加算される閾値ベクトルのサイズを表したタプル。この層の次元情報から自動的に初期化される。

- self.weight

  入力に掛けられる重み行列を表す変数。このクラスでは単位行列に初期化される。

- self.shift

  入力に加算される閾値ベクトルを表す変数。このクラスではゼロベクトルに初期化される。

- self.activator

  重み付け後に適応される活性関数をあらわす変数。このクラスでは恒等変換に初期化される。

重要なメソッドは以下の通りである。

- calculate(self)

  Layer クラスのメソッドをオーバーライドしている。まず前の層の calculate を行い、その最新の出力を入力として受け取る。その入力に self.weight をかけ、self.shift を加算することで重み付けを行う。最後に活性関数 self.activator を適応し得られた値を最新の出力として更新する。また、更新後の値を返す。この再帰的な calculate の呼び出しにより、ニューラルネットワークの最後の層の calculate を呼び出すことでネットワーク全ての値を順方向に更新できることになる。

- setWeight(self, weight)

  重み行列 self.weight を設定する。この層に合わない形の行列が入力された場合には設定は行われず、警告文が標準出力に表示される。

- setShift(self, shift)

  閾値ベクトル self.shift を設定する。この層に合わない形のベクトルが入力された場合には設定は行われず、警告文が標準出力に表示される。

- setActivator(self, function)

  活性化関数 self.activator を設定する。関数型以外の値が入力された場合には設定は行われず、警告文が標準出力に表示される。

1.3.4 HiddenLayer

中間層を表すクラス。変換を行う層であるので、ConversionLayer を継承する。

ConversionLayer との違いは、その初期化の内容である。こちらのクラスでは、コンストラクタで 2 つの乱数シード weightSeed と shiftSeed を要求し

（デフォルトの値は 1 ）、その乱数シードを使って生成したランダム値に基づいて重み行列 self.weight と閾値ベクトル self.shift の初期値が設定される。また、活性化関数 self.activator は**シグモイド関数**に初期化される。

1.3.5 OutputLayer

出力層を表すクラス。変換を行う層であるので、ConversionLayer を継承する。

ConversionLayer との違いは、その初期化の内容である。こちらのクラスでは、コンストラクタで 2 つの乱数シード weightSeed と shiftSeed を要求し（デフォルトの値は 1 ）、その乱数シードを使って生成したランダム値に基づいて重み行列 self.weight と閾値ベクトル self.shift の初期値が設定される。また、活性化関数 self.activator は**ソフトマックス関数**に初期化される。

2 実行結果

「>>」以下が入力、「Recognition Result :」以下が出力である。なお、今後の開発のことを考慮し、その他のデータも一部表示する仕様にした。

実行例を以下に示す。

```
...neural_network> python task1.py
### task1 ###
start loading (testing data)
finish loading
select number.
>> 1
1 is selected.

Recognition Result : 6
(likelihood : 0.113937661163)

[[ 0.10316129]
 [ 0.10057583]
 [ 0.09511654]
 [ 0.10266565]
 [ 0.09758801]
 [ 0.08950398]
 [ 0.11393766]
 [ 0.09026591]
 [ 0.11292604]
```

```
 [ 0.09425909]]

continue?
select yes or no.
>> yes
select number.
>> 200
200 is selected.

Recognition Result : 8
(likelihood : 0.109248250636)

[[ 0.1038663 ]
 [ 0.10188789]
 [ 0.0975031 ]
 [ 0.09667546]
 [ 0.10470781]
 [ 0.09053544]
 [ 0.10627111]
 [ 0.09216106]
 [ 0.10924825]
 [ 0.09714358]]

continue?
select yes or no.
>> no
Bye.
```

3 工夫

- 全てのプログラムはオブジェクト指向・ドメインモデル方式で設計されている。これによりモジュールの再利用・拡張が容易になり、今後様々なニューラルネットを組み上げることができる。
- 入出力を任意の回数行えるようなインターフェースとした。
- 乱数シード、層の次元数等の定数は容易に変更が可能となるように設計した。

4　問題点

　他の人より進捗がやや遅い。綺麗な設計でも素早く作れるようになりたいものである。