計算機科学実験及演習 4
画像認識 第２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: 2018 年 1 月 11 日
提出日: 2018 年 1 月 10 日

[課題 2] ミニバッチ対応＆クロスエントロピー誤差の計算

[課題 1] のコードをベースに，ミニバッチ（＝複数枚の画像）を入力可能とするように改良し，さらにクロスエントロピー誤差を計算するプログラムを作成せよ．

- MNIST のテスト画像 10000 枚の中からランダムに B 枚をミニバッチとして取り出すこと．
- クロスエントロピー誤差の平均を標準出力に出力すること．
- ニューラルネットワークの構造，重みは課題 1 と同じでよい．
- バッチサイズ B は自由に決めて良い（100 程度がちょうどよい）．
- ミニバッチを取り出す処理はランダムに行う

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
 ┣ nn.py                … 組み合わせ済ニューラルネットのモジュール
```

```
┃  ┗ NeuralNetwork3Layers   … 3 層ニューラルネットのクラス
┣ 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
┗ task2.py                 … 課題 2 プログラム
MNIST
```

課題 1 から追加したのは、nn.py モジュールと task2.py プログラムである。nn.py モジュールでは、課題 1 で構築した 3 層ニューラルネットワークとその機能を NeuralNetwork3Layers クラスに集約した。task2.py は課題 2 のメインプログラムである。

また、ミニバッチの入力に対応し、クロスエントロピーの計算をできるようにするため、既存のモジュールの各クラスにも必要なメソッドを追加した。

1.2 メインプログラム

課題 2 のメインプログラム test2.py は以下の通りである。入力はニューラルネットに入力するバッチの内容を確定するための乱数シード、出力はニューラルネットの出力から算出されたクロスエントロピー誤差である。追加の機能として、入出力を任意の回数行えるようにした。

test1.py

```
1    import ioex (以下 import 省略)
5
6    BATCH_QUANTITY = 100
7
8    print("### task2 ###")
9
10   print("batch mode")
11
12   # 入出力準備
13   inputM = ioex.InputManager()
14   outputM = ioex.OutputManager()
15   testingData = inputM.getMnistTestingData()
16   testingData.shuffle()
17
18   neuralNet = nn.NeuralNetwork3Layers(28*28, 50, 10)
19
20   loop = True
21   while(loop):
22       print("Input random seed number of batch.")
23       begin = inputM.selectNumber()
24       imageBatch = testingData
                .getImageBatch(BATCH_QUANTITY, begin)
25       answerBatch = testingData
                .getAnswerVecotrBatch(BATCH_QUANTITY, begin)
26       neuralNet.setInputBatch(imageBatch)
27       neuralNet.setAnswerBatch(answerBatch)
28
29       print("start")
30       result = neuralNet.calculate()
31       #print(result)
32       totalLoss = neuralNet.getLoss()
33
34       print("\nResult.")
35       print("totalLoss : " + str(totalLoss))
36
37       print("\ncontinue?")
38       loop = inputM.selectYesOrNo()
39
40   print("Bye.")
```

16 行目では、MNIST データ群の順番をランダムに変更するメソッド MnistDataBox#shuffle を呼び出している。これは後の課題で複数回バッチデータによる学習を行う際に、入力バッチデータを変更するためのメソッドである。18 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。NeuralNetwork3Layers クラスのコンストラクタの第 1 引数は InputLayer の次元数、第 2 引数は HiddenLayer の次元数、第 3 引数は OutputLayer の次元数である。各 Layer 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が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 24,25 行目の MnistDataBox# getImageBatch メソッド,MnistDataBox#getAnswerVecotrBatch メソッド によって入力バッチとそれに対応するラベルバッチを取得する。いずれのメソッドも第 1 引数は取得するバッチサイズ、第 2 引数はバッチ取得の際に使用する基準位置を指定する。次に、26,27 行目の NeuralNetwork3Layers#setInputBatch メソッド,NeuralNetwork3Layers#setAnswerBatch メソッドによって入力バッチとラベルバッチをニューラルネットにセットする。続いて、30 行目の NeuralNetwork3Layers#calculate メソッドで入力バッチに基づいてニューラルネット内の計算を行う。最後に 32 行目の NeuralNetwork3Layers#getLoss メソッドによって 出力とラベルバッチからクロスエントロピー誤差を計算している。

1.3 nn モジュール

構成済みのニューラルネットワークを提供するモジュール。メインプログラムではこの nn モジュール内のクラスより下位のモジュールやクラス（layer モジュール等）を認知、操作する必要はないようモジュール作成をした。

1.3.1 NeuralNetwork3Layers クラス

構成済み 3 層ニューラルネットワークのクラス。Layer モジュールのメソッドを隠蔽する。また、ラベルデータの保持、管理を行っている。

nn.py

```
class NeuralNetwork3Layers :
   def __init__(self, inputDim, hiddenDim, outputDim):
       # 三層ニューラルネットワーク
       self.inputLayer = layer.InputLayer(inputDim)
       self.hiddenLayer =
         layer.HiddenLayer(hiddenDim,self.inputLayer)
       self.outputLayer =
         layer.OutputLayer(outputDim,self.hiddenLayer)
       # 正解データ
       self.answer = np.zeros(outputDim)

   ## 入力系メソッド ###

   def setInput(self, inputData):
       self.inputLayer.setInput(inputData)

   def setInputBatch(self, inputBatch):
       self.inputLayer.setInputBatch(inputBatch)

   def setAnswer(self, answerVector):
       self.answer =
         answerVector.reshape(answerVector.size,1)
       if(self.answer.size != self.outputLayer.dimension):
           print("WARNING!
             Input data is NOT match to deimension")

   def setAnswerBatch(self,answerBatch):
       self.answer = answerBatch

   ## 活性メソッド ##

   def calculate(self):
       return self.outputLayer.calculate()

   def getLoss(self):
       return self.outputLayer.getLoss(self.answer)
```

主なメソッドは以下の通りである。

- __init__(self, inputDim, hiddenDim, outputDim)

  インスタンスを作成し値を初期化する。３つの層に含まれるニューロンの数 (次元数) をここで決定する。

- setInput(self, inputData)

  単一入力をニューラルネットにセットする。

- setInputBatch(self, inputBatch)

  バッチ入力をニューラルネットにセットする。

- setAnswer(self, answerVector)

  単一ラベルデータをニューラルネットにセットする。

- setAnswerBatch(self,answerBatch)

  バッチラベルデータをニューラルネットにセットする。

- calculate(self)

  現在セットされている入力に基づいてニューラルネットを活性化させ順方向伝播を行う。

- getLoss(self)

  現在の出力とセットされているラベルデータに基づいてクロスエントロピー誤差平均を算出する。

1.4 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。

1.4.1 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。

課題２から追加されたインスタンス変数、メソッドは以下の通りである。

- self.lossFunction

  誤差計算に使用する関数を格納するインスタンス変数。初期設定ではクロスエントロピー誤差関数がセットされている。

- getLoss(self, answer)

  現在の出力値と引数に指定したラベルデータを元に、self.lossFunction に格納された関数を使用して計算を行い、誤差の値を返す。

その他のインスタンス変数、メソッド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。

- __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。

1.4.2 InputLayer クラス

入力層を表すクラス。Layer クラスを継承する。
課題 2 から追加されたメソッドは以下の通りである。

- setInputBatch(self, inputBatch)

  バッチ入力をニューラルネットにセットする。

その他のインスタンス変数、メソッド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。

- 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。

1.4.3 ConversionLayer クラス

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。

主なインスタンス変数、メソッド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。

- 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。

1.4.4 HiddenLayer クラス

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。

1.4.5 OutputLayer クラス

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。

1.5 data モジュール

処理の対象となるデータを表現したクラスを集めたモジュール。

1.5.1 MnistDataBox クラス

MNIST データセットを表すクラス。外部仕様としては後に記す MnistData クラスの集合であるが、内部仕様としては単一 MNIST データを取り出す時に MnistData クラスを生成する。また、バッチの生成も行う。
主なメソッドは以下の通りである。

- getSingleData(self, num)

  インデックスを引数で指定し、単一の MNIST データを表す MnistData オブジェクトを取得する。

- shuffle(self, seed = 1)

  内部的な MNIST データの保持順、インデックスをシャッフルする。単一 MNIST データやバッチを取り出す際に、シャッフルの前後ではインデックスが同一であっても別のデータを取り出すことができる。

- getImageBatch(self, batchSize, shift = 0)

  画像データをバッチ形式で取り出す。引数にはバッチサイズとバッチを取り出す基準となるインデックスを指定する。

- getAnswerVecotrBatch(self, batchSize, shift = 0)

  ラベルデータをバッチ形式で取り出す。引数にはバッチサイズとバッチを取り出す基準となるインデックスを指定する。

- getSize(self)

  MNIST データセットのサイズ（格納している MNIST データの数）を返す。

1.5.2 MnistData クラス

MNIST 単一データを表すクラス。
主なメソッドは以下の通り。

- getImage(self)

  画像データを行列形式で取り出す。

- getAnswer(self)

  ラベルデータを取り出す。

- getAnswerAsVector(self, size = None)

  ラベルデータを、そのラベルに対応するインデックスが 1 、それ以外のインデックスが 0 のベクトル形式で取り出す。

2　実行結果

「>>」以下が入力、「Result. TotalLoss :」以下が出力である。
実行例を以下に示す。

```
....neural_network> python task2.py
### task2 ###
batch mode
start loading (testing data)
finish loading
Input random seed number of batch.
select number.
>> 0
0 is selected.
start

Result.
totalLoss : 2.30539980393

continue?
select yes or no.
>> yes
Input random seed number of batch.
select number.
>> 4
4 is selected.
start

Result.
totalLoss : 2.30575362662

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。

- 各レイヤー内で使用される関数を分離している。これにより、容易に使用される関数を変更することができ、幅広い種類のニューラルネットを簡単に構成できる。
- 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。
- バッチは元データの行列をスライスして作成している。バッチ作成に for ループを使用しないことで高速化を図った。

4 問題点

一般化を心がける余り、コードがやや肥大化してしまっている。応用課題に取り組む時まで一般化の恩恵は受けられないので、効果が今のところ実感できない。