計算機科学実験４レポート3 必修課題

工学部情報学科3回生 計算機科学コース
1029310587 谷 綜一郎

2022 年1 月11 日

目 次

第1章　課題内容

1.1　概要

[課題 2] のコードをベースに、3 層ニューラルネットワークのパラメータ $W^{(1)}$, $W^{(2)}$, $\boldsymbol{b}^{(1)}$, $\boldsymbol{b}^{(2)}$ を学習するプログラムを作成する。

1.2　諸条件

- ネットワークの構造、バッチサイズは課題 2 と同じで良い。
- 学習には MNIST の学習データ 60000 枚を用いること。
- 繰り返し回数は自由に決めて良い。教師データの数を N、ミニバッチのサイズを B としたとき、N/B 回の繰り返しを 1 **エポック**と呼び，通常はエポック数とミニバッチサイズで繰り返し回数を指定する。
- 学習率 η は自由に決めて良い。（0.01 あたりが経験的に良さそう）
- 各エポックの処理が終了する毎に、クロスエントロピー誤差を標準出力に出力すること。（これにより学習がうまく進んでいるかどうかを確認することができる）
- 学習終了時に学習したパラメータをファイルに保存する機能を用意すること。フォーマットは自由。パラメータを numpy の配列 (ndarray) で実装している場合は numpy.save() や numpy.savez() 関数が利用できる。
- ファイルに保存したパラメータを読み込み、再利用できる機能を用意すること。（numpy.load() 関数が利用できる）

第2章　プログラム説明

2.1　概要

program3.py で用いた誤差逆伝播法について記述した後、program2.py からの変更点・追加点を説明する。

2.2　誤差逆伝播法

重み $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ を修正するために、損失関数 E_n の勾配（$\frac{\partial E_n}{\partial W^{(1)}}$ など）を誤差逆伝播法により計算する。勾配が計算できれば、重みの修正を

$$W_{(1)} \leftarrow W_{(1)} - \eta \frac{\partial E_n}{\partial W^{(1)}} \tag{1}$$

のように更新することができる。（η は後述する**学習率**である）

2.2.1　連鎖律

ある合成関数 $f(g(x))$ が与えられた場合、x に対する f の偏微分 $\frac{\partial f}{\partial x}$ は、

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} \tag{2}$$

によって与えられる。

３層ニューラルネットワークにおいて、入力から出力までの処理は順に、入力信号⇒全結合層 $W^{(1)}, \boldsymbol{b^{(1)}}$ ⇒シグモイド関数⇒全結合層　$W^{(2)}, \boldsymbol{b^{(2)}}$ ⇒ Softmax 関数⇒損失関数 E_n の順に処理される。この一連の処理を合成関数であると考えれば、勾配 $\frac{\partial En}{\partial W^{(1)}}$ や $\frac{\partial En}{\partial W^{(2)}}$ の計算を、Softmax 関数の出力に対する損失関数の偏微分、Softmax 関数の偏微分、全結合層の偏微分、... を順に計算することによって実現することができる。

2.2.2　Softmax 関数+クロスエントロピー誤差の逆伝播

Softmax 関数層は C 個の入力 $\boldsymbol{a} = (a_1, ..., a_C)^T$ を受け取り、C 個の出力 $y_k^{(2)}(k = 1, ..., C)$ を返し、クロスエントロピー誤差 E は $y_k^{(2)}$ を受け取り、誤差 E を出力する関数である。勾配 $\frac{\partial En}{\partial a_k}$ は、次式で与えられる（証明は割愛）。

$$\frac{\partial En}{\partial a_k} = y_k^{(2)} - y_k \tag{3}$$

また、E_n は E の平均であるため、$\frac{\partial E_n}{\partial E} = \frac{1}{B}$（$B$ はバッチサイズ）となり、勾配 $\frac{\partial E_n}{\partial a_k}$ は、

$$\frac{\partial E_n}{\partial a_k} = \frac{y_k^{(2)} - y_k}{B} \tag{4}$$

となる。

2.2.3 全結合層の逆伝播

ベクトル $\boldsymbol{x}$ を入力とし、行列 W、ベクトル $\boldsymbol{b}$ をパラメータとして、線形和 $\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b}$ を出力する関数を考える。$\frac{\partial E_n}{\partial \boldsymbol{y}}$ を用いて、

$$\frac{\partial E_n}{\partial \boldsymbol{x}} = W^T \frac{\partial E_n}{\partial \boldsymbol{y}} \tag{5}$$

$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial \boldsymbol{y}} \boldsymbol{x}^T \tag{6}$$

$$\frac{\partial E_n}{\partial \boldsymbol{b}} = \frac{\partial E_n}{\partial \boldsymbol{y}} \tag{7}$$

となる（証明は割愛）。ミニバッチの場合、すなわち B 個のベクトル $\boldsymbol{x_1}, ..., \boldsymbol{x_B}$ とそれに対する出力 $\boldsymbol{y_1}, ..., \boldsymbol{y_B}$、および出力に対する損失関数の勾配 $\frac{\partial E_N}{\partial \boldsymbol{y_i}} (i = 1, ..., B)$ が与えられた場合、W や $\boldsymbol{b}$ に対する損失関数の勾配は、B 個のベクトル $\boldsymbol{x_1}, ..., \boldsymbol{x_B}$ を各列に持つ行列を X、$\frac{\partial E_N}{\partial \boldsymbol{y_i}}$ を各列に持つ行列を $\frac{\partial E_N}{\partial Y}$ として、

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \tag{8}$$

$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial Y} X^T \tag{9}$$

$$\frac{\partial E_n}{\partial \boldsymbol{b}} = rowSum(\frac{\partial E_n}{\partial Y}) \tag{10}$$

で与えられる。ここで、$rowsum$ は行列における行ごとの和を表す。

2.2.4 シグモイド関数の逆伝播

シグモイド関数 $a(t) = 1/(1 + exp(-t))$ の微分は、

$$a(t)' = (1 - a(t))a(t) \tag{11}$$

で与えられる。シグモイド関数への入力を x、出力を y とし、$\frac{\partial E_n}{\partial y}$ が与えられたとき、$\frac{\partial E_n}{\partial x}$ は、

$$\frac{\partial E_n}{\partial x} = \frac{\partial y}{\partial x} \frac{\partial E_n}{\partial y} = \frac{\partial E_n}{\partial y}(1 - y)y \tag{12}$$

で与えられる。

2.2.5 処理手順

1. ミニバッチを作成
2. ミニバッチに対する出力を順伝播によって計算
3. 損失関数の値 E_n を算出
4. 式 4 を用いて $\frac{\partial E_n}{\partial a_k}$ を計算。$\frac{\partial E_n}{\partial a_k}$ は全部で $B \times C$ 個得られる。
5. 式 8，9，10 を用いて $\frac{\partial E_n}{\partial X}, \frac{\partial E_n}{\partial W}, \frac{\partial E_n}{\partial \boldsymbol{b}}$ を計算。式 8，9，10 内の $\frac{\partial E_n}{\partial Y}$ として、前の手順で得られた $\frac{\partial E_n}{\partial a_k}$ を用いる。ここで計算した $\frac{\partial E_n}{\partial W}, \frac{\partial E_n}{\partial \boldsymbol{b}}$ が、それぞれ $\frac{\partial E_n}{\partial W^{(2)}}, \frac{\partial E_n}{\partial \boldsymbol{b^{(2)}}}$ となる。
6. 式 12 を $\frac{\partial En}{\partial X}$ の各要素に適用
7. 式 8，9，10 を用いて $\frac{\partial E_n}{\partial X}, \frac{\partial E_n}{\partial W}, \frac{\partial E_n}{\partial \boldsymbol{b}}$ を計算。式 8，9，10 内の $\frac{\partial E_n}{\partial Y}$ として、前の手順で得られたシグモイド関数の微分値を用いる。ここで計算した $\frac{\partial E_n}{\partial W}, \frac{\partial E_n}{\partial \boldsymbol{b}}$ が、それぞれ, $\frac{\partial E_n}{\partial W^{(1)}}, \frac{\partial E_n}{\partial \boldsymbol{b^{(1)}}}$ となる。
8. 学習率 η を用いて、パラメータを更新。

$$W_{(1)} \leftarrow W_{(1)} - \eta \frac{\partial E_n}{\partial W^{(1)}} \tag{13}$$

$$W_{(2)} \leftarrow W_{(2)} - \eta \frac{\partial E_n}{\partial W^{(2)}} \tag{14}$$

$$\boldsymbol{b_{(1)}} \leftarrow \boldsymbol{b_{(1)}} - \eta \frac{\partial E_n}{\partial \boldsymbol{b_{(1)}}} \tag{15}$$

$$\boldsymbol{b_{(2)}} \leftarrow \boldsymbol{b_{(2)}} - \eta \frac{\partial E_n}{\partial \boldsymbol{b_{(2)}}} \tag{16}$$

2.3 外部仕様

課題内容で記述した通り、program3.py は 3 層ニューラルネットワークにおける 4 つのパラメータを学習するプログラムである。以下に、簡単な操作手順を示す。

1. あらかじめ保存したファイルがあるかどうかを y/n で入力する。y の場合は、続けて名前を入力することでそのファイルを読み込み、n の場合は、[課題 2] と同様に、パラメータの初期値をランダムに設定する。
2. 教師データを述べすべての枚数学習するサイクルを 1 エポックとして、50 エポック学習する。この際、1 エポック内での平均クロスエントロピー誤差と、教師データにおける平均一致率が表示される。
3. 最終的に学習したパラメータをファイルに保存するかどうかを y/n で入力する。y の場合は、続けて名前を入力することでファイルに保存し、n の場合は、Done と表示してプログラムを終了する。

2.4 内部設計

2.4.1 概要

課題 2 で実装した諸関数に、必要な関数を適宜追加・変更することで program3.py 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 $main3()$ を、以下のプログラム 2.1 に示す。

プログラム 2.1: main3()

```
1  def main3():
2    (w1, b1, w2, b2) = loadOrTemplate()
3    total = 0.
4    accuracy = 0
5    for i in range(Epoc):
6        for j in range(int(ImageNum/BatchSize)):
7            (img, ans) = preprocess()
8            prop = propagation(img, ans, w1, b1, w2, b2)
9            ((w1, b1, w2, b2), max, e) = backPropagation(w1, b1, w2, b2, prop)
10           total = total + e
11           accuracy = accuracy + np.count_nonzero(ans == max)/BatchSize
12       print(str(i+1) + "_epoc : " + str(total/(ImageNum/BatchSize)))
13       total = 0.
14       print(str(i+1) + "_result: " + str(int((accuracy/(ImageNum/BatchSize))
             *100))+ "\%")
15       accuracy = 0
16   save((w1, b1, w2, b2))
```

このコード中の変数については以下のとおりである。

- img：100 × 28 × 28 の画像データ（3 次元配列）
- ans：正解ラベルデータ 100 個（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 × 100 の画像データ
- y1：100 × 100 の二次元配列
- a：10 × 100 の二次元配列
- y2：10 × 100 の二次元配列
- max：y2 のうち最大の数値のインデックス 100 個（0〜9 の整数値）
- e：クロスエントロピー誤差

- prop：関数 propagation における x, y1, y2, onehot, e, max をまとめた組
- total： 1 エポックにおけるクロスエントロピー誤差の総計
- accuracy： 1 エポックにおける画像認識一致率の総計

2.4.2 グローバル変数

- X：60000 × 28 × 28 の画像データ（3 次元配列）
- Y：60000 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：画像教師データの枚数（ここでは 60000）
- 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）
- BatchSize：バッチサイズ（ここでは 100）
- LearnPara：学習率（ここでは 0.01）
- Epoc：エポック数（ここでは 50）

2.4.3 変更した関数

- crossEntropy

具体的なコードの変更点は、以下のプログラム 2.2 に示すとおりである。クロスエントロピー誤差 $e2$ に加えて、onehot ベクトルである $onehot$ を合わせた組 e を返すように変更した。

プログラム 2.2: crossEntropy

```
1   def crossEntropy(answer, out):
2    onehot = np.identity(10)[answer]
3    e1 = np.dot(-onehot, np.log2(out))
4    e2 = (np.sum(np.diag(e1)))/BatchSize
5    e = onehot, e2
6    return e
```

- propagation

具体的なコードの変更点は、以下のプログラム 2.3 に示すとおりである。上記の crossEntropy の変更に合わせて、変数に $onehot$ を追加した。また、この関数は $x, y1, y2, onehot, e, max$ を合わせた組 $prop$ を返す。

プログラム 2.3: propagation

```
def propagation(img, ans, w1, b1, w2, b2):
 x = inputLayer(img)
 y1 = sigmoid(combine(x, w1, b1))
 a = combine(y1, w2, b2)
 y2 = softmax(a)
 max = np.argmax(y2, axis=0)
 (onehot, e) = crossEntropy(ans, y2)
 prop = (x, y1, y2, onehot, e, max)
 return prop
```

2.4.4 追加した関数

- revEntropy

  具体的な新しいコードは、以下のプログラム 2.4 に示すとおりである。処理手順の 4 に基づいてクロスエントロピー誤差の逆伝播を考える。

プログラム 2.4: revEntropy

```
def revEntropy(y2, onehot):
 return (y2 - onehot.T) / BatchSize
```

- revCombine

  具体的な新しいコードは、以下のプログラム 2.5 に示すとおりである。処理手順の 5, 7 に基づいて結合層の逆伝播を考える。

プログラム 2.5: revCombine

```
def revCombine(w, x, enY):
 enX = np.dot(w.T, enY)
 enW = np.dot(enY, x.T)
 enB = np.sum(enY, axis=1).reshape(-1,1)
 en = enX, enW, enB
 return en
```

- revSigmoid

  具体的な新しいコードは、以下のプログラム 2.6 に示すとおりである。処理手順の 6 に基づいてシグモイド関数の逆伝播を考える。

プログラム 2.6: revSigmoid

```
1  def revSigmoid(y, enY):
2   return enY*(1-y)*y
```

- renew

具体的な新しいコードは、以下のプログラム 2.7 に示すとおりである。処理手順の 8 に基づいてパラメータを更新する。

プログラム 2.7: renew

```
1  def renew(w1, b1, w2, b2, enW1, enW2, enB1, enB2):
2   N = LearnPara
3   newW1 = w1 - N*enW1
4   newB1 = b1 - N*enB1
5   newW2 = w2 - N*enW2
6   newB2 = b2 - N*enB2
7   new = newW1, newB1, newW2, newB2
8   return new
```

- backPropagation

具体的な新しいコードは、以下のプログラム 2.8 に示すとおりである。これは、上記の関数を用いた逆伝播を行う関数である。$w1, b1, w2, b2, prop$ を入力として、学習した新しいパラメータ new、$prop$ に含まれている max, e を出力とする。

プログラム 2.8: backPropagation

```
1  def backPropagation(w1, b1, w2, b2, prop):
2   (x, y1, y2, onehot, e, max) = prop
3   enY2 = revEntropy(y2, onehot)
4   (enX2, enW2, enB2) = revCombine(w2, y1, enY2)
5   enY1 = revSigmoid(y1, enX2)
6   (enX1, enW1, enB1) = revCombine(w1, x, enY1)
7   new = renew(w1, b1, w2, b2, enW1, enW2, enB1, enB2)
8   return new, max, e
```

このコードにある変数の説明は以下の通りである。

- 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 × 100 の画像データ

- y1：100 × 100 の二次元配列
- y2：10 × 100 の二次元配列
- max：y2 のうち最大の数値のインデックス 100 個（0〜9 の整数値）
- e：クロスエントロピー誤差
- prop：関数 propagation における x, y1, y2, onehot, e, max をまとめた組
- enY2：$\frac{\partial E_n}{\partial a_k}$ を表す
- enX2：$\frac{\partial E_n}{\partial y1}$ を表す
- enW2：$\frac{\partial E_n}{\partial W^{(2)}}$ を表す
- enB2：$\frac{\partial E_n}{\partial \boldsymbol{b}^{(2)}}$ を表す
- enY1：combine(x, w1, b1) を a' としたとき、$\frac{\partial E_n}{\partial a'}$ を表す
- enX1：今回この変数は使っていない
- enW1：$\frac{\partial E_n}{\partial W^{(1)}}$ を表す
- enB1：$\frac{\partial E_n}{\partial \boldsymbol{b}^{(1)}}$ を表す

- loadOrTemplate

具体的な新しいコードは、以下のプログラム 2.9 に示すとおりである。これは保存したファイルがあるかどうか標準入力から確認するための関数である。ynL に標準入力から受け取った y, n を格納する。y の場合は、続けて名前を入力することでそのファイルを読み込み、n の場合は、[課題 2] と同様に、パラメータの初期値をランダムに設定する。

プログラム 2.9: loadOrTemplate

```
1   def loadOrTemplate():
2    ynL = input("Do you have a file? y/n ")
3    if ynL == "y":
4        name = input("What's your file's name? ('.npy' isn't needed!) ")
5        try:
6            return np.load(name + ".npy")
7        except:
8            print("Not found!")
9            return loadOrTemplate()
10   elif ynL == "n":
11       return preWB()
12   else:
13       print("Please 'y' or 'n'.")
14       return loadOrTemplate()
```

- save

具体的な新しいコードは、以下のプログラム 2.10 に示すとおりである。これは学習したパラメータを保存するかどうか標準入力から確認するための関数である。ynS に標準入力から

受け取った y, n を格納する。y の場合は、続けて名前を入力することでその名前のファイルに保存して、n の場合は、Done と表示してプログラムを終了する。

プログラム 2.10: save

```
1  def save(data):
2   ynS = input("Do you save file? y/n ")
3   if ynS == "y":
4       name = input("Please enter the file's name. ('.npy' isn't needed
            !) ")
5       print("Saved.")
6       return np.save(name, data)
7   elif ynS == "n":
8       print("Done.")
9   else:
10      print("Please enter 'y' or 'n'.")
11      return save(data)
```

第3章　実行結果

実行結果を以下 3.1 に示す。なお、この結果は保存したファイルを使わず、初期パラメータをランダムで設定した場合であり、最終的に学習したパラメータは保存していない。

プログラム 3.1: 実行結果

```
>python program3.py
Do you have a file? y/n n
1_epoc : 1.3216802127748075
1_result: 79\%
2_epoc : 0.6637866863934347
2_result: 89\%
3_epoc : 0.5317283595044244
3_result: 91\%
4_epoc : 0.4613842836017639
4_result: 91\%
5_epoc : 0.4235936796031284
5_result: 92\%
..
46_epoc : 0.16126398849077073
46_result: 96\%
47_epoc : 0.1603151868805586
47_result: 97\%
48_epoc : 0.15387208948225523
48_result: 97\%
49_epoc : 0.15091910128629107
49_result: 97\%
50_epoc : 0.15281057132786838
50_result: 97\%
Do you save file? y/n n
```

第4章　工夫点・問題点

loadOrTemplate, save

課題 1 の inputNumber 同様に、適切でない入力にはエラーメッセージを返すことで、プログラムが終了しないようにしている。

main3

当初、「(img, ans) = preprocess()」を j ループではなく i ループに記述していたため、以下の 4.1 に示すようにクロスエントロピー誤差の増減に違和感があった。当初のままでは、教師データのごく一部を何周も学習する仕様になっていたことが原因と考えられたので、教師データをのべ全て学習できるように j ループに移動した。

プログラム 4.1: 当初の出力結果

```
..
13_epoc : 0.2950703574867874
13_result: 96\%
14_epoc : 0.27851366277419637
14_result: 96\%
15_epoc : 0.22615694616498117
15_result: 96\%
16_epoc : 0.26319333088649616
16_result: 97\%
17_epoc : 0.19232569045082357
17_result: 96\%
18_epoc : 0.1742842994219058
18_result: 98\%
19_epoc : 0.18105245388450647
19_result: 97\%
20_epoc : 0.27358853288502
20_result: 94\%
..
```