課題3レポート

情報学科 計算機コース 1029275871 芦田聖太

提出日　18/1/19

課題 3

1　誤差逆伝播による 3 層ニューラルネットークの学習

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 W1, W2, b1, b2 を学習プログラムを作成する。

仕様

- 学習には MNIST の学習データ 60000 枚を使用する。
- バッチサイズは 100。
- 繰り返し回数、学習率は自由に設定する。
- 各エポック終了毎にクロスエントロピー誤差を標準出力に出力する。
- 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。

2　設計方針

- 定数の設定
- バッチの選択
- バックプロパゲーション
- エポック終了時のクロスエントロピーの表示
- 学習終了時のパラメータのファイルへの書き込み

以上の 5 つの項目に関して、前回の課題 2 から変更を加えた。3 層のニューラルネットワークの構成や、入力データに関しては前回から変更は加えていない。

3　実装とプログラムの説明

3.1　定数の設定

ニューラルネットに関しての定数はこれまでと変わっていないので、学習に関しての定数だけ説明を加えておく。loop は学習回数で、len(X) / batch で 1 エポックの学習回数 600 回を表しているので、学習回数は 30 エポックである。percent は学習率を表しており、0.01 にしている。

```
# 学習
loop = int((len(X) / batch) * 30)
percent = 0.01
```

3.2 バッチの選択

60000 枚の画像データの前から順に 100 枚ずつ選び学習データとして使用する。場合分けを行なっているのは、繰り返し回数が 600 に達したときにうまく学習データを取り出すためである。

main2.py

```
if ((n + 1) * batch) % 60000 != 0:
    learn = np.reshape(X[(n * batch) % 60000: ((n + 1) * batch) % 60000:], (batch,
        row * row)).T
    answer = Y[(n * batch) % 60000: ((n + 1) * batch) % 60000:]
else:
    learn = np.reshape(X[(n * batch) % 60000: 60000:], (batch, row * row)).T
    answer = Y[(n * batch) % 60000: 60000:]
```

3.3 バックプロパゲーション

- クロスエントロピーとソフトマックスの逆伝播
  a_k をソフトマックス関数の C 個の入力の k 番目の要素、$y_k^{(2)}$ はソフトマックス関数の C 個の出力の k 番目の要素である。クロスエントロピー誤差は $y_k^{(2)}$ を入力にとり E を返す。また、y_k は one-hot vector 表記の解答の k 番目の要素であり、B はバッチサイズを表している。En は E の平均である。

$$\frac{\partial E_n}{\partial a_k} = \frac{y_k^{(2)} - y_k}{B} \tag{1}$$

- 全結合層の逆伝播
  ベクトル x を入力として行列 W とベクトル b とする。このとき $y = Wx + b$ を出力する関数を考える。B 個のベクトル $x_1, x_2, ...x_B$ を各列に持つ行列を X とする。また、$x_1, x_2, ...x_B$ に対する出力である B 個のベクトルをそれぞれ $y_1, y_2, ...y_B$ とするとき、$\frac{\partial E_n}{\partial y_i}$ を各列に持つ行列を $\frac{\partial E_n}{\partial Y}$ とする。

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \tag{2}$$

$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial Y} X^T \tag{3}$$

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \tag{4}$$

- シグモイド関数の逆伝播
  シグモイド関数を a で表すと、シグモイド関数自体の微分は以下のようになる。

$$a(t)' = (1 - a(t))a(t) \tag{5}$$

  シグモイド関数への入力ベクトルのうちの 1 つの要素を t とすると、逆伝播は以下の式で表される。

$$\frac{\partial E_n}{\partial t} = \frac{\partial E_n}{\partial a(t)} \frac{\partial a(t)}{\partial t} \tag{6}$$

  実際には、B 個のベクトル $x_1, x_2, ...x_B$ を各列に持つ行列を X が入力となるのでこの計算は各要素毎に行われることになる。

以上の 3 つの項目を実現したのが以下のコードとなる。逆伝播 1 というのが、クロスエントロピー+ソフトマックスの逆伝播と中間層と出力層の間の全結合の逆伝播を表してる。次に逆伝播 2 というのが、シグモイド関数の逆伝播と入力層と中間層の全結合の逆伝播を表している。
answer：解答、finout：ソフトマックス関数の出力、batch：バッチサイズ、midout：シグモイド関数の出力、learn：入力行列、weight1 と weight2：重み、end：出力層の数、middle：中間層の数

```
    # 逆伝播1
    aen_ay2 = (finout - np.eye(end)[answer].T) / batch
    aen_ax2 = weight2.T.dot(aen_ay2)
    aen_aw2 = aen_ay2.dot(midout.T)
    aen_ab2 = np.reshape(np.sum(aen_ay2, axis=1), (end, 1))

    # 逆伝播2
    aen_ay1 = aen_ax2 * ((1 - midout) * midout)
    aen_ax1 = weight1.T.dot(aen_ay1)
    aen_aw1 = aen_ay1.dot(learn.T)
    aen_ab1 = np.reshape(np.sum(aen_ay1, axis=1), (middle, 1))

    # 重み修正
    weight1 -= percent * aen_aw1
    b1 -= percent * aen_ab1
    weight2 -= percent * aen_aw2
    b2 -= percent * aen_ab2
```

3.4 クロスエントロピーの表示

n は学習回数を表しており、60000 の学習データを網羅する毎に、表示するように設定している。

```
# クロスエントロピー
    entropy = funcs.cross(finout, answer, end)
    if n * batch % 60000 == 0:
        print(str(n) + "回目")
        print(entropy)
```

3.5 パラメータの保存

np.savez を使い、すべてのパラメータを 1 つのファイルに保存した。また、np.savetxt を用いて保存されているパラメータを確認できるようにした。

```
# パラメータの保存
print("save")
np.savez("parameters.npz", w1=weight1, w2=weight2, b1=b1, b2=b2)
np.savetxt('weight1.csv', weight1, delimiter=',')
np.savetxt('weight2.csv', weight2, delimiter=',')
```

4 実行結果

実行結果を以下に示す。1 回の学習で 100 のデータを扱うため、600 回でちょうど 1 エポックとなる。最初、クロスエントロピーの平均値は約 2.23 となるが 600 回の学習で 0.255 まで抑えることができている。そこから多少の上下はあるが、着実に誤差を減らすことができており 30 エポックで約 0.050 付近で落ち着いた。

```
0 回目
2.29753145973

600 回目
0.255184237804

1200 回目
0.194213906587

1800 回目
0.168603437976
```

```
2400 回目
0.157386061456

3000 回目
0.130301105324

3600 回目
0.120298183322

4200 回目
0.111763262959

4800 回目
0.101924064509

5400 回目
0.0973306610888

6000 回目
0.102313912085

6600 回目
0.0909826454486

7200 回目
0.0924376689691

7800 回目
0.0808173999405

8400 回目
0.080460494789

9000 回目
0.0803265560274

9600 回目
0.0826512117024

10200 回目
0.0689447874985

10800 回目
0.0695081780207

11400 回目
0.0777434063435

12000 回目
0.0629360379799

12600 回目
0.0670838369799

13200 回目
0.0632502479807

13800 回目
0.0672145942555

14400 回目
0.0584590187365

15000 回目
0.0559522828233

15600 回目
0.0588288039049

16200 回目
```

```
0.047666561674

16800 回目
0.051013089824

17400 回目
0.0499271202031
```

5 考察

5.1 工夫点

過学習を避けるために学習回数を 30 エポックにした。30 エポックで学習した結果、テスト画像の正答率が 97.2% で、50 エポックで学習した結果が 97.37% であった。正答率がほぼ変わらないのであれば汎用性が高い方が良いと考えたため、30 エポックのパラメータを保存した。

5.2 問題点

3 層ニューラルネットワークをテキスト通り作り、コンテストに出してみたが 64.84%とあまり高い値は出なかった。発展課題である畳み込み層の設計を行うとかなり認識度が上がるようなので、取り組んでみたいと思う。

6 コード全文

```
import funcs
import numpy as np
from mnist import MNIST

mndata = MNIST("/Users/omushota/ex4-image/le4nn")
X, Y = mndata.load_training()
X = np.array(X)
X = X.reshape((X.shape[0], 28, 28))
Y = np.array(Y)

# 定数 ##############################
# 入力データ関連
line = X.shape[0]
row = X.shape[1]

# バッチ
batch = 100

# 学習
loop = int((len(X) / batch) * 30)
percent = 0.01

# 重み1
middle = 300
average = 0
variance1 = 1.0 / (row * row)
seed = 1
np.random.seed(seed)

weight1 = np.random.normal(average, variance1, (row * row * middle))
weight1 = np.reshape(weight1, (middle, row * row))
b1 = np.random.normal(average, variance1, middle)
b1 = np.reshape(b1, (middle, 1))
```

```
# 重み2
end = 10
variance2 = 1.0 / middle

weight2 = np.random.normal(average, variance2, middle * end)
weight2 = np.reshape(weight2, (end, middle))
b2 = np.random.normal(average, variance2, end)
b2 = np.reshape(b2, (end, 1))

# 傾き
aen_ay2 = np.zeros((end, batch))
aen_ax2 = np.zeros((middle, batch))
aen_aw2 = np.zeros((end, middle))
aen_ab2 = np.zeros((end, 1))

aen_ay1 = np.zeros((middle, batch))
aen_ax1 = np.zeros((row*row, batch))
aen_aw1 = np.zeros((middle, row*row))
aen_ab1 = np.zeros((middle, 1))


# 学習 ###################################
for n in range(loop):
    # バッチ選択
    if ((n + 1) * batch) % 60000 != 0:
        learn = np.reshape(X[(n * batch) % 60000: ((n + 1) * batch) % 60000:], (batch,
            row * row)).T
        answer = Y[(n * batch) % 60000: ((n + 1) * batch) % 60000:]
    else:
        learn = np.reshape(X[(n * batch) % 60000: 60000:], (batch, row * row)).T
        answer = Y[(n * batch) % 60000: 60000:]

    # 中間層
    midin = weight1.dot(learn) + b1
    midout = funcs.sigmoid(midin)

    # 出力層
    finin = weight2.dot(midout) + b2
    finout = funcs.softmax(finin)

    # クロスエントロピー
    entropy = funcs.cross(finout, answer, end)
    if n * batch % 60000 == 0:
        print(str(n) + "␣回目")
        print(entropy)

    # 逆伝播1
    aen_ay2 = (finout - np.eye(end)[answer].T) / batch
    aen_ax2 = weight2.T.dot(aen_ay2)
    aen_aw2 = aen_ay2.dot(midout.T)
    aen_ab2 = np.reshape(np.sum(aen_ay2, axis=1), (end, 1))

    # 逆伝播2
    aen_ay1 = aen_ax2 * ((1 - midout) * midout)
    aen_ax1 = weight1.T.dot(aen_ay1)
    aen_aw1 = aen_ay1.dot(learn.T)
    aen_ab1 = np.reshape(np.sum(aen_ay1, axis=1), (middle, 1))

    # 重み修正
    weight1 -= percent * aen_aw1
    b1 -= percent * aen_ab1
    weight2 -= percent * aen_aw2
    b2 -= percent * aen_ab2


# パラメータの保存
print("save")
np.savez("parameters.npz", w1=weight1, w2=weight2, b1=b1, b2=b2)
np.savetxt('weight1.csv', weight1, delimiter=',')
np.savetxt('weight2.csv', weight2, delimiter=',')
```