計算機科学実験及演習 4 画像認識
レポート 3

高橋駿一　2018 年度入学 1029-30-3949

提出日: 2020 年 10 月 30 日

課題 3

課題内容

[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を学習するプログラムを作成した. なお, 本レポートは [課題 2] からの差分について記述した.

作成したプログラムの説明

誤差逆伝播法を実装したため, 課題 2 の時点と比較して関数の数が大幅に増加したため, 既存の関数を機能別でクラスにまとめることで可読性と再利用性を高めた.

ソースコード 1: パラメータの設定

```
1   class params:
2       def __init__(self, M, d):
3           np.random.seed(seed=32)
4           self.W = np.random.normal(0, 1/d, (d, M))
5           self.b = np.random.normal(0, 1/d, (1, M))
6           self.eta = 0.01
7
8       def update(self, dW, db):
9           self.W −= self.eta ∗ dW
10          self.b −= self.eta ∗ db
11
12      def save(self, i):
13          np.save('./w{}'.format(i), self.W)
14          np.save('./b{}'.format(i), self.b)
```

パラメータに関するクラスである. __init__() でインスタンス化を行い, 重み W と切片ベクトル b の初期化を行う. なお, 学習率 eta は 0.01 とした. update() で W, b の更新を行い, save() でこれらを.npy ファイルとして保存する.

ソースコード 2: ファイルに保存したパラメータの読み込み

```
1   def load(i):
2       W_loaded = np.load('./w{}.npy'.format(i))
3       b_loaded = np.load('./b{}.npy'.format(i))
4       return W_loaded, b_loaded
```

.npy ファイルに保存したパラメータを読み込む関数である. 引数 i で読み込むファイルを制御している.

ソースコード 3: 線形和の計算とその逆伝播

```
class matrix_operation:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.X = None

    def forward(self, X):
        self.X = X
        y = np.dot(X, self.W) + self.b
        return y

    def backward(self, back):
        dX = np.dot(back, self.W.T)
        dW = np.dot(self.X.T, back)
        db = np.sum(back, axis=0)
        return dX, dW, db
```

線形和に関するクラスである．forward() で線形和の計算を行い, backward() でその逆伝播の計算を行う．

ソースコード 4: シグモイド関数の計算とその逆伝播

```
class sigmoid:
    def __init__(self):
        self.y = None

    def forward(self, t):
        self.y = (1 / (1 + np.exp(−1 ∗ t)))
        return self.y

    def backward(self, back):
        dt = back ∗ (1 − self.y) ∗ self.y
        return dt
```

シグモイド関数についてのクラスである．forward() でシグモイド関数の適用を行い, backward() でその逆伝播の計算を行う．

ソースコード 5: ソフトマックス関数の計算とその逆伝播

```
class softmax:
    def __init__(self, batch_size):
        self.y_pred = None
        self.batch_size = batch_size

    def forward(self, a):
        alpha = np.tile(np.amax(a, axis=1), 10).reshape(10, self.batch_size).T
        # print('max', alpha)
        exp_a = np.exp(a − alpha)
        # print('e', exp_a)
        sum_exp = np.tile(np.sum(exp_a, axis=1), 10).reshape(10, self.batch_size).T
        # print('sum', sum_exp)
        self.y_pred = exp_a / sum_exp
```

```
14                  return self.y_pred
15
16              def backward(self, y_ans, B):
17                  da = (self.y_pred − y_ans) / B
18                  return da
```

ソフトマックス関数についてのクラスである． forward() でソフトマックス関数の適用を行い, backward() でその逆伝播の計算を行う. なお, 課題 2 のレポートでは alpha を計算する際に axis を指定していなかったため, alpha が行列 a の成分の中で最も大きい値を取っていたが, 本来は上記のコードのように行列 a の各行の中で最も大きい値を取り, それを行列 a と同じサイズに拡張した行列となる.

ソースコード 6: ニューラルネットワーク

```
1       class neural_network():
2           def __init__(self, batch_size, epoch, middle_layer, last):
3               self.batch_size = batch_size
4               self.epoch = epoch
5               self.middle_layer = middle_layer
6               self.last = last
7
8           def learning(self):
9               params1 = params(self.middle_layer, 784)
10              params2 = params(self.last, self.middle_layer)
11              for i in range(self.epoch):
12                  loss = []
13                  for j in range(int(60000 / self.batch_size)):
14                      input_vec, image_size, batch_index, class_sum = input_layer2(train_X, j)
15                      batch_label = train_Y[batch_index[j]]
16                      y_ans = np.identity(10)[batch_label]
17
18                      W1, b1 = params1.W, params1.b
19                      mo1 = matrix_operation(W1, b1)
20                      t = mo1.forward(input_vec)
21                      # print('matrix', t)
22                      sig = sigmoid()
23                      y1 = sig.forward(t)
24                      # print('sigmoid', y1)
25                      W2, b2 = params2.W, params2.b
26                      mo2 = matrix_operation(W2, b2)
27                      a = mo2.forward(y1)
28                      # print('a', a)
29                      soft = softmax(self.batch_size)
30                      y2 = soft.forward(a)
31                      # print(y2)
32                      # binary_y = postprocessing(y2)
33                      # print(binary_y)
34                      E = cross_entropy_loss(y2, y_ans)
35                      loss.append(E)
36
37                      da = soft.backward(y_ans, self.batch_size)
38                      dX2, dW2, db2 = mo2.backward(da)
39                      dt = sig.backward(dX2)
40                      dX1, dW1, db1 = mo1.backward(dt)
41                      params1.update(dW1, db1)
42                      params2.update(dW2, db2)
43
44                  print(np.sum(loss) / len(loss))
45
46              params1.save(1)
47              params2.save(2)
```

```
48
49          def testing(self):
50              input_vector, image_size, i, class_num = input_layer(test_X)
51              # y_ans = np.identity(10)[test_Y[i]]
52              W1, b1 = load(1)
53              mo1 = matrix_operation(W1, b1)
54              t = mo1.forward(input_vector)
55              # print('matrix', y1)
56              sig = sigmoid()
57              y1 = sig.forward(t)
58              # print('sigmoid', y1)
59              W2, b2 = load(2)
60              mo2 = matrix_operation(W2, b2)
61              a = mo2.forward(y1)
62              # print('a', a)
63              soft = softmax(1)
64              y2 = soft.forward(a)
65              # print(y2)
66              binary_y = postprocessing(y2)
67              print(np.where(binary_y == 1)[1][0], test_Y[i])
```

ここまでに作成したクラスや関数を用いてニューラルネットワークを構築する関数である. __init__() でインスタンス化を行い, バッチサイズ batch_size, エポック epoch, 中間層のノード数 middle_layer, 出力層のノード数 (クラス数)last を指定する. learning() では 1 エポックを 60000/batch_size 回の繰り返しとし, パラメータの更新を実行する. testing() では保存された.npy ファイルから W1, W2, b1, b2 を読み込み, これらのパラメータを使ってテストデータの画像認識を行い, ニューラルネットワークの計算結果と正解のラベルを標準出力に出力する.

ソースコード 7: 課題 3 の実行

```
1   nn = neural_network(100, 100, 50, 10)
2   print('学習を開始します.␣')
3   nn.learning()
4   print('テストを開始します.␣')
5   nn.testing()
```

neural_network クラスをインスタンス化し, 課題 3 を実行する.

実行結果

実行の結果, クロスエントロピー誤差が 2.296782644875204 から 0.2190637709925514 に減少し, 重み W1, W2 と切片ベクトル b1, b2 のファイルが作成された. さらに, これらのファイルは正常に読み込まれた.

工夫点

- この後の課題でも活用しやすくするために関数を機能別でクラスに集約し, 実装した.
- softmax 関数で行列に関する演算を実装する際に for 文を使わず, numpy の機能を活用することで計算時間を抑えた.

問題点

- 課題 2 の時と比べるとクラスを活用することで改善されたことではあるが, バッチサイズを一箇所で管理できておらず, create_batch(X) 内の batch_size とは別で input_layer2(X) 内で input_vector を取得するためにバッチサイズである 100 をそのまま記述してしまっている. また, neural_network クラスでも改めて指定している.