画像実験課題 3

1029323422 天野岳洋

2023 年 1 月 19 日

1　概要

課題 2 からコードを改良し, パラメータを学習するプログラムを作成する.

2　設計の方針

基本的には課題 2 の内容に加え定義通り逆伝播層を導入し, 学習率に基づいて更新を行う. それを何度か繰り返しを行えばよい. ただし, 伝播層と逆伝播層では行列の形が違うことに注意する.

3　実装

前準備

三層 NN で必要になる変数の準備を行う. 実装の具体的なコードは以下のとおりである.

Listing 1　前準備

```
1  import numpy as np
2  import mnist
3
4  seed, M, C, B = 601, 80, 10, 100
5  #study_rate
6  my = 0.01
7  np.random.seed(seed)
8  X = mnist.download_and_parse_mnist_file("train-images-idx3-ubyte.gz")
9  Y = mnist.download_and_parse_mnist_file("train-labels-idx1-ubyte.gz")
10 img_size = X[0].size
11 epoch = X.shape[0] // B
12 #epoch_repeat
13 num = 30
14
15 W1 = np.random.normal(loc = 0, scale = np.sqrt(1/img_size), size=(M,
       img_size))
16 b1 = np.random.normal(loc = 0, scale = np.sqrt(1/img_size), size=(M, 1))
17 W2 = np.random.normal(loc = 0, scale = np.sqrt(1/M), size=(C, M))
18 b2 = np.random.normal(loc = 0, scale = np.sqrt(1/M), size=(C, 1))
```

課題 2 以前で説明した内容については省略する. 新しく導入したものとして, my, epoch, num がある. my は学習率であり, 教科書通り 0.01 としている. epoch は 1epoch で何回繰り返すかを表しており, データ数//バッチサイズによって求めている. num は何 epoch 繰り返すかを表しており, これは適当な数字を用いている.

ロード

パラメータの初期化方法として, ランダム初期化以外に, ロードによる初期化も必要である. これを可能にするコードは以下のとおりである.

Listing 2　load

```
print("random -> 0, load -> 1")
a = int(input())
if(a == 1):
  parameters = np.load("./Parameters/parameter.npz")
  W1 = parameters['arr_0']
  W2 = parameters['arr_1']
  b1 = parameters['arr_2']
  b2 = parameters['arr_3']
```

内容については容易であるため, 省略する.

学習

ここではまとめて三層 NN のパラメータを更新する手続きについて説明を行う. ただし, back-propagation についての説明は後のセクションで詳しく行う. 実装の具体的なコードは以下のとおりである.

Listing 3　3NN

```
for i in range(num):
  for j in range(epoch):
      batch_random = np.random.randint(0, 60000, B)
      before_conv = np.array(X[batch_random])
      img = before_conv.reshape((B, img_size, 1))
      answer = np.array(Y[batch_random])
      #one-hot vectorize
      onehot = np.zeros((answer.size, 10))
      onehot[np.arange(answer.size), answer] = 1
      onehot = onehot.reshape(B, C, 1)


      #propagation
      input1 = np.matmul(W1, img) + b1
      output1 = 1/(1 + np.exp(-1 * input1))
      input2 = np.matmul(W2, output1) + b2
      alpha = np.repeat(input2.max(axis = 1), C, axis= 1).reshape(B, C, 1)
```

```
18          sumexp = np.repeat(np.sum(np.exp(input2 - alpha), axis=1), 10, axis
                =1).reshape(B, C, 1)
19          output_last = np.exp(input2 - alpha) / sumexp
20          crossE += (-1/B)* np.sum(onehot * np.log(output_last))
21
22          #back
23          delta_b = ((output_last - onehot)/B).reshape(B, C).T
24
25          delta_y1 = np.dot(W2.T, delta_b)
26          delta_W2 = np.dot(delta_b, output1.reshape(B, M))
27          delta_b2 = np.sum(delta_b, axis = 1).reshape(C, 1)
28
29          delta_a = delta_y1 * (1 - output1.reshape(B, M).T) * output1.reshape(
                B, M).T
30
31          delta_x = np.matmul(W1.T, delta_a)
32          delta_W1 = np.matmul(delta_a, img.reshape(B, img_size))
33          delta_b1 = np.sum(delta_a, axis=1).reshape(M, 1)
34
35          #update
36          W1 = W1 - my * delta_W1
37          b1 = b1 - my * delta_b1
38          W2 = W2 - my * delta_W2
39          b2 = b2 - my * delta_b2
40      print(crossE / epoch)
41  print("finish")
42  np.savez("parameter", W1, W2, b1, b2)
```

まず初めに説明した数だけの繰り返しを行っている. その後, 課題 2 でやった通りの方法で, 画像の適切な形への変更, one-hotvecor の作成, 順伝播を行っている. そして微分を計算して, それに学習率をかけた分だけそれぞれのパラメータから引いている.

後処理としては, 1epoch が終わるたびに crossEntropy を出力, すべての epoch が終われば, finish という文字列を出力し, パラメータを parameter というファイルに保存している.

逆伝播層

ここでは, 逆伝播層についての詳しい説明を行う.

ソフトマックス層

まずは形を確認する. ソフトマックス層に置ける入力の形は (B, C, 1), 出力の形も同じく (B, C, 1) である. したがって, 入力についての偏微分は, $y_{ik1}^{(2)}$ を 3NN のバッチ内の i 番目の画像に対する

数字 k に関する最終的な出力, y_{ik1} を onehot_vector のバッチ内 i 番目の数字 k に関する値であるとすると,

$$\frac{\partial E_n}{\partial a_{ik1}} = \frac{y^{(2)}_{ik1} - y_{ik1}}{B}$$

によって求められ, この左辺の形は (B, C, 1) である.

ここでは次の全結合層で求められる形が (C, B) であることから, あらかじめその形に変形しておく. その説明は次のセクションで行う. 以上の手続きを行うコードは以下のとおりである.

Listing 4　SoftMax

```
1  delta_b = ((output_last - onehot)/B).reshape(B, C).T
```

全結合層

全結合層においては, 出力の偏微分が与えられており, それに基づいて, 入力, 重み, バイアスについての偏微分を行うものとする. この理由として, 全結合層が最後の出力層となることは考えにくいので, 一つ出力層に近い層の入力に関する微分値が与えられているはずだからである.

では詳しい計算の手続きを述べる. 出力の偏微分は,$\boldsymbol{y_i}$ をミニバッチ内 i 番目の出力 (=後ろの層の入力) であるとし、$\partial E_N/\partial \boldsymbol{y_i}$ を各列に持つ行列を $\partial E_N/\partial Y$ として与えられているものとする. つまり, $\partial E_N/\partial Y$ は (?, B) という形をしているものである. ?に入る数字は二つ目の全結合層であるならば, C であり, 一つ目の全結合層ならば, M である. この形に注意を払えば後は教科書通りであり,

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \tag{1}$$

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

$$\frac{\partial E_n}{\partial \boldsymbol{b}} = rowSum(\frac{\partial E_n}{\partial Y} X^T) \tag{3}$$

によって, それぞれの微分値が求められる. ただし, $\partial E_n/\partial X$ の行列の形は入力 (B, X, 1) とは異なり, (X, B) という形になっていることに注意したい. 実際のコードは以下の通りである.

Listing 5　全結合層 2

```
1  delta_y1 = np.dot(W2.T, delta_b)
2  delta_W2 = np.dot(delta_b, output1.reshape(B, M))
3  delta_b2 = np.sum(delta_b, axis = 1).reshape(C, 1)
```

ここでは, 伝播層では入力を (B, M, 1) として与えていたものを, (B, M) に変更しているほか, delta_b2 の形を引き算ができるように, (C,) から (C, 1) へと変更している. また, delta_b は前の節で得たものであり, 適切な形へと変更されている. もう一つの全結合層については説明を省略する. 実際のコードは以下のとおりである.

Listing 6　全結合層 1

```
delta_x = np.matmul(W1.T, delta_a)
delta_W1 = np.matmul(delta_a, img.reshape(B, img_size))
delta_b1 = np.sum(delta_a, axis=1).reshape(M, 1)
```

シグモイド関数

シグモイド関数への入力を x_{ik1} とし, 出力を y_{ik1} として, $\partial E_n/\partial y_{ki}$ が与えられたときに,

$$\frac{\partial E_n}{\partial x_{ki}} = \frac{\partial y_{ki}}{\partial x_{ki}} \frac{\partial E_n}{\partial y_{ki}} = \frac{\partial E_n}{\partial y_{ki}}(1 - y_{ki})y_{ki}$$

これを実行するコードは以下のとおりである.

Listing 7　シグモイド関数

```
delta_a = delta_y1 * (1-output1.reshape(B, M).T) * output1.reshape(B, M).T
```

4　実際の動作

実際に実行すると以下のような標準出力が得られた.

```
1epoch: 100\%
train loss2.1195802822692515
validation loss1.9059151233838645
2epoch: 100\%
train loss1.6942129635495269
validation loss1.4607649183991045
3epoch: 100\%
train loss1.2908205032287965
validation loss1.1183198557225995
```

さらに, parameter.npz というファイルが保存されていることを確認した. ここで表示されている validation loss や 100% については工夫点のセクションで説明を行う.

4.1　学習の様子

学習の様子をテストデータ, トレーニングデータそれぞれに対しての正解率, 又はクロスエントロピー誤差について注目することによって調べた. まず初めに少ないエポック数のグラフを乗せる.

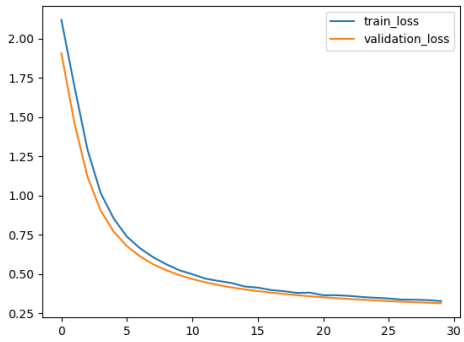

図 1　mini_loss

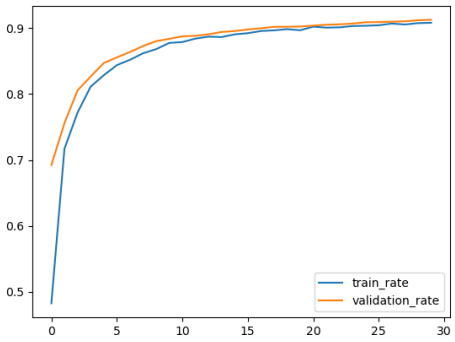

図 2　mini_rate

このグラフを見れば, うまく学習が進んでいることがわかる. テストデータに対しての方が成績が良くなっているのは, 訓練データに対して学習させたのちに, テストデータに対してパフォーマンスを図っているためである. この次にエポック数を 10000 にして収束を図る. それが次のグラフである.

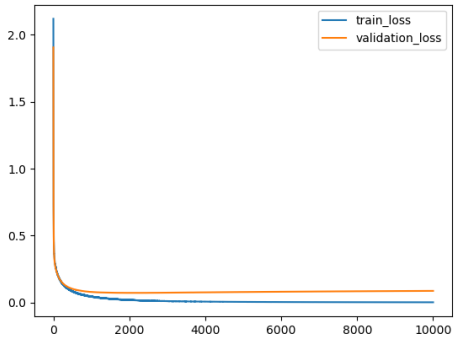

図 3　mini_loss

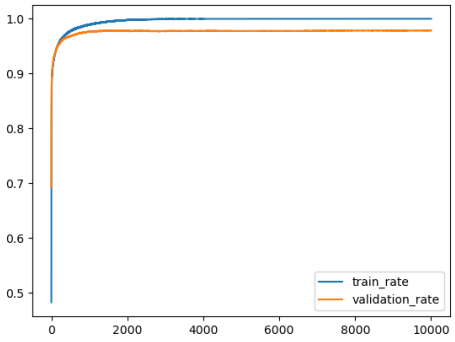

図 4　mini_rate

上の二つのグラフを見れば, 最終的に訓練データに対しての正答率は 100% に限りなく近い値になっていることがわかる. またその際のクロスエントロピー誤差は 0.0016 となった. 一方でテストデータに対しての正答率は 0.95% を目安にとどまった.

5　工夫点

ロード機能を実装することによって, 途中からの学習が可能になった. また mnist の test データを validation-data とし, そのデータに対するクロスエントロピーも表示するようにした.

さらに for 文を try 文に入れ, 後に except KeybordInterrupt をキャッチすることによって, 学習を途中で打ち切って保存することが可能になった. これはもっとも簡単な過学習を防ぐ方法である early-stopping を行うことができるようになったことを意味する.

また今現在何エポック目の何パーセントまで完了しておくか表示する機能も実装した. この機能の具体的な活用方法は思いつかないが, 学習が正しく行われていることを改めて確認できる.

6 問題点

ロードする場合に, そのロード先のファイルが実際に存在しない場合にエラーを出力してしまう. また, early-stopping ができるようになったとは言うが, 手動で行わなければならずずっと監視しておく必要がある.