計算機科学実験4 画像処理
課題3 レポート

1029311501 坂井優斗

提出日：2022年1月13日

1　課題内容

課題 2 のコードをベースとして, 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。プログラムの仕様は以下の通りである。

- 学習用データは MNIST の学習データ 60000 枚を利用する。
- ニューラルネットワークの構造は前回の課題 2 と同様のものとする。
- バッチサイズは $B = 100$, エポック数は 10 とする。
- 学習率は, $\eta = 0.01$ とする。
- 各エポックの学習が終了次第, 1 エポックに含まれる各バッチに対する損失の平均を出力する。
- 学習終了時にパラメータを保存できるようにする。また, 保存したパラメータを読み込んで利用できるようにする。

2　プログラムの説明

課題 3 の内容は task3.py というソースコードで実装している。ここでは, 前課題から修正を加えたり, 追加したコードについて説明をまとめる。本レポートで触れられていないプログラム部分については課題 1, 課題 2 についてまとめたレポートを参照していただきたい。

2.1　グローバル変数

- parameters : 辞書型オブジェクト. key を $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$, value をパラメータの値として, ニューラルネットワークのパラメータの値を保持している。
- learning_rate : 学習率 η

2.2　plot_figure

Code 1: plot_figure

```
1    def plot_figure(list):
2      plt.figure(figsize=(8, 6))
3      plt.plot(np.arange(1, len(list) + 1), list)
4      plt.xlabel('iteration')
5      plt.ylabel('loss')
6      plt.title('Cross␣Entropy␣Error', fontsize=20)
7      plt.grid()
8      plt.show()
```

この関数は Matplotlib を利用してグラフを描くための関数である。具体的には, パラメータ学習時のクロスエントロピー誤差の推移を横軸にイテレーション, 縦軸に損失値として表示する。
引数 list にクロスエントロピー誤差を記録しているリストを渡す。

2.3 init_parameters

Code 2: init_parameters

```
1  def init_parameters():
2    parameters["W_1"] = np.random.normal(loc=0, scale=np.sqrt(1/input_node_size), size=(
       inner_node_size, input_node_size))
3    parameters["b_1"] = np.random.normal(loc=0, scale=np.sqrt(1/input_node_size), size=(
       inner_node_size, 1))
4    parameters["W_2"] = np.random.normal(loc=0, scale=np.sqrt(1/inner_node_size), size=(
       output_node_size, inner_node_size))
5    parameters["b_2"] = np.random.normal(loc=0, scale=np.sqrt(1/inner_node_size), size=(
       output_node_size, 1))
```

この関数は, パラメータ $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ を初期化する関数である。パラメータは学習前段階で値を持っていないので, 乱数値で適当に初期化する。乱数は課題 1 と同じ仕様で生成している。

2.4 load_parameters

Code 3: load_parameters

```
1  def load_parameters(file_name):
2    load_param = np.load(file_name)
3    parameters["W_1"] = load_param["W_1"]
4    parameters["W_2"] = load_param["W_2"]
5    parameters["b_1"] = load_param["b_1"]
6    parameters["b_2"] = load_param["b_2"]
```

この関数は, パラメータ $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ に外部ファイルから値を与える関数である。引数 file_name に, numpy.savez() 関数を利用してパラメータの情報が保存されているファイル名を渡す。そして, numpy.load 関数でそのファイルからパラメータの値を読み込んで, 変数に代入する。

2.5 calc_derivative_softmax

Code 4: calc_derivative_softmax

```
1  def calc_derivative_softmax(processed_images, labels):
2    derivative_softmax_list = []
3    for i in range(batch_size):
4      processed_image, label = processed_images[i].reshape(-1), labels[i]
5      label_vector = np.zeros(output_node_size)
6      label_vector[label] = 1
7      derivative_softmax_list.append(list((processed_image - label_vector)/batch_size))
8    return derivative_softmax_list
```

この関数は, クロスエントロピー誤差 E_n をソフトマックス関数の入力 $\boldsymbol{x}$ で微分した勾配 ∇E_n を求める関数である。
ここで, ソフトマックス関数の入力を $\boldsymbol{x} = (x_1, x_2, \ldots, x_c)$, 出力を $\boldsymbol{y} = (y_1, y_2, \ldots, y_c)$, ラベルの値に関する one-hot-vector を $\boldsymbol{y'}$ とすると,

$$\frac{\partial E_n}{\partial x_k} = \frac{y_k - y'_k}{B} \tag{1}$$

で与えられる。6 行目で one-hot-vector をつくり, 7 行目で (1) 式の計算をしている。また, バッチの各画像に対する勾配を連結したリスト derivative_softmax_list を返すようにしている。

2.6 calc_derivative_sigmoid

Code 5: calc_derivative_sigmoid

```
1   def calc_derivative_sigmoid(y, derivative_y):
2     derivative_sigmoid_list = []
3     for i in range(batch_size):
4       derivative_y_i = (derivative_y.T[i]).reshape(inner_node_size, 1)
5       derivative_sigmoid_list.append(list(derivative_y.T[i] * y[i] * (1-y[i])))
6     return derivative_sigmoid_list
```

この関数は, クロスエントロピー誤差 E_n をシグモイド関数の入力 $\boldsymbol{x}$ で微分した勾配 ∇E_n を求める関数である。
ここで, シグモイド関数の入力を $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, 出力を $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ とすると,

$$\frac{\partial E_n}{\partial x_k} = \frac{\partial E_n}{y_k}(1 - y_k)y_k \tag{2}$$

で与えられる。5 行目で (2) 式の計算を行い, calc_derivative_softmax と同じようにバッチの各画像に対する勾配を連結して, derivative_softmax_list として返している。

2.7 main 関数

Code 6: main 関数

```
1   def main():
2     load_image()
3
4     choice = input("use␣parameter␣file␣?␣[yes/no]:␣")
5     if choice == "yes":
6       file_name =  input("input␣the␣parameter␣file␣name:␣")
7       load_parameters(file_name)
8     elif choice == "no":
9       init_parameters()
10
11    training_loss_list = []
12
13    for i in tqdm(range(1, epoch+1)):
14      with tqdm(total=(len(train_images)//batch_size), leave=False) as pbar:
15        for j in range(len(train_images)//batch_size):
16          pbar.set_description('Epoch␣{}'.format(i))
17          batchs, labels = get_batch()
18          x = input_layer(batchs)
19          y = inner_layer(x)
20          processed_batchs = output_layer(y)
21
22          derivative_a = np.array(calc_derivative_softmax(processed_batchs, labels))
23          derivative_X_2 = np.dot(parameters["W_2"].T, derivative_a.T)
24          derivative_W_2 = np.dot(derivative_a.T, y)
25          derivative_b_2 = (np.sum(derivative_a.T, axis=1)).reshape(output_node_size, 1)
26          derivative_t = np.array(calc_derivative_sigmoid(y, derivative_X_2))
27          derivative_X_1 = np.dot(parameters["W_1"].T, derivative_t.T)
28          derivative_W_1 = np.dot(derivative_t.T, x)
29          derivative_b_1 = (np.sum(derivative_t.T, axis=1)).reshape(inner_node_size, 1)
30
31          parameters["W_1"] -= (learning_rate*derivative_W_1)
32          parameters["W_2"] -= (learning_rate*derivative_W_2)
33          parameters["b_1"] -= (learning_rate*derivative_b_1)
34          parameters["b_2"] -= (learning_rate*derivative_b_2)
35
36          cross_entropy_error = loss_function(processed_batchs, labels)
37          training_loss_list.append(cross_entropy_error)
38          pbar.update(1)
39
40      cross_entropy_error_mean = np.mean(training_loss_list[(i-1)*(len(train_images)//
          batch_size):len(training_loss_list)-1])
41      tqdm.write(f"The␣loss␣in␣epoch{i}␣is␣{cross_entropy_error_mean}.")
42
```

```
43      if(input("save␣the␣parameter␣?␣[yes/no]:␣") == "yes"):
44        np.savez('parameters_task3.npz', W_1=parameters["W_1"], W_2=parameters["W_2"], b_1=
            parameters["b_1"], b_2=parameters["b_2"])
45
46      plot_figure(training_loss_list)
```

課題 2 から増えた処理について説明を加える。

- 4 行目でファイルで用意した学習済みのパラメータを利用するかどうかを”yes”または”no”で入力し, yes ならばその後ファイル名を入力して load_parameters 関数でパラメータの読み込み, no ならば init_parameters 関数でパラメータの初期化を行う。

- 13 行目から学習を行なっている。エポックの数分のループの中で, 学習用イメージの数をバッチサイズで割った値の数分のループを回して, 各バッチに対してパラメータの更新をしている。22 行目から 29 行目で $\frac{\partial E_n}{\partial W^{(1)}}, \frac{\partial E_n}{\partial W^{(2)}}, \frac{\partial E_n}{\partial \boldsymbol{b}^{(1)}}, \frac{\partial E_n}{\partial \boldsymbol{b}^{(2)}}$ を計算している。2.5 calc_derivative_softmax, 2.6 _calc_derivative_sigmoid, またミニバッチを利用した全結合層における逆伝播が

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \tag{3}$$
$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial Y} X^T \tag{4}$$
$$\frac{\partial E_n}{\partial \boldsymbol{b}} = rowsum\left(\frac{\partial E_n}{\partial Y}\right) \tag{5}$$

  で与えられることを利用している。

- 31 行目から 34 行目で以下の (6) 式から (9) 式を使って, パラメータの更新を行なっている。

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial E_n}{\partial W^{(1)}} \tag{6}$$
$$W^{(2)} \leftarrow W^{(2)} - \eta \frac{\partial E_n}{\partial W^{(2)}} \tag{7}$$
$$\boldsymbol{b^{(1)}} \leftarrow \boldsymbol{b^{(1)}} - \eta \frac{\partial E_n}{\partial \boldsymbol{b^{(1)}}} \tag{8}$$
$$\boldsymbol{b^{(2)}} \leftarrow \boldsymbol{b^{(2)}} - \eta \frac{\partial E_n}{\partial \boldsymbol{b^{(2)}}} \tag{9}$$

- 37 行目で各バッチの損失を記録するためのリストである training_loss_list にクロスエントロピー誤差の値を入れている。

- 41 行目で各エポックのクロスエントロピー誤差を出力する。

- 43 行目, 44 行目で学習したパラメータを保存するかどうかを”yes”または”no”で入力し, yes を受け取ったら numpy.savez 関数でパラメータを保存する。

- 最後に 46 行目で損失の値の推移を, 2.2 plot_figure で描画している。

4 行目でファイルで用意した学習済みのパラメータを利用するかどうかを”yes”または”no”で入力し, yes ならばその後ファイル名を入力して load_parameters 関数でパラメータの読み込み, no ならば init_parameters 関数でパラメータの初期化を行う。

2.8　その他

課題 2 で実装したプログラムにおける, input_layer, inner_layer, output_layer は map 関数を利用して複数枚の画像に対する処理が定義されていたが, 行列演算で処理することで高速化に成功している。

```
  def input_layer(images):
    images = np.array(images)
    trans_images = np.reshape(images, (batch_size, input_node_size))
    return trans_images

  def inner_layer(images):
    affine = (np.dot(parameters["W_1"], images.T) + parameters["b_1"]).T
    return sigmoid_function(affine)

  def output_layer(images):
    affine = (np.dot(parameters["W_2"], images.T) + parameters["b_2"]).T
    return softmax_function(affine)
```

3　実行結果

コマンド python task3.py でスクリプトを実行する。
標準出力から, 損失の値が段々と減少していることが確認できた。また, 以下の図 1 からはじめのエポックで損失の値は急激に減少し, その後は振動しながらゆるやかに減少してる様子がわかる。

ターミナル

```
$ python task2.py
  use parameter file ? [yes/no]: no
  The loss in epoch1 is 0.9786762972115907.
  The loss in epoch2 is 0.47267874016283723.
  The loss in epoch3 is 0.37425864869892156.
  The loss in epoch4 is 0.32922751341709156.
  The loss in epoch5 is 0.29778827033033795.
  The loss in epoch6 is 0.27326234763971663.
  The loss in epoch7 is 0.25758884261443765.
  The loss in epoch8 is 0.24507068128321882.
  The loss in epoch9 is 0.23506002517117333.
  The loss in epoch10 is 0.2205426182961792.
  save the parameter ? [yes/no]: no
```

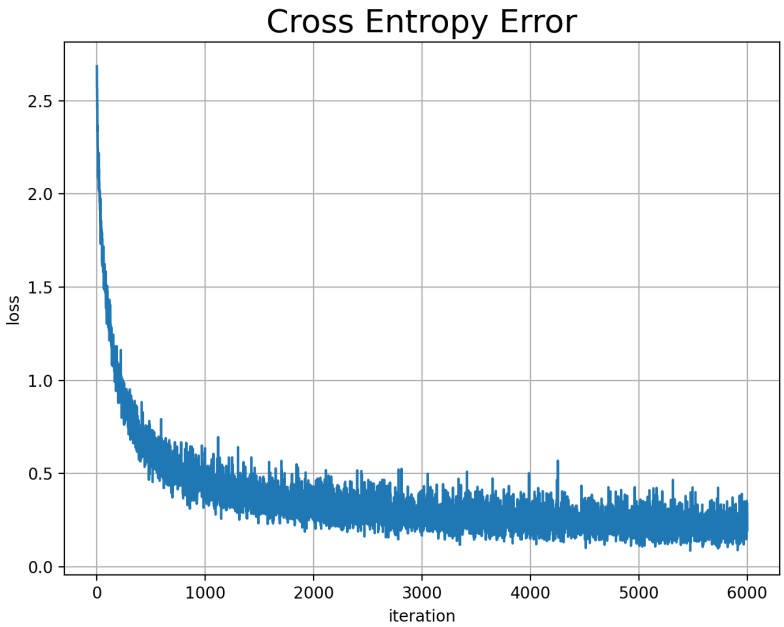

図 1: クロスエントロピー誤差の推移

4 工夫点

- tqdm というライブラリを利用して, 学習状況をプログレスバーとして表示して視覚的に進捗を確認できるようにした。その様子を以下の図 2 に示す。

```
(base) yutonoMacBook-Air:isle4_image_recognition yutorse$ python task3.py
use parameter file ? [yes/no]: no
The loss in epoch1 is 0.9518018490825925.
The loss in epoch2 is 0.4827033228928052.
 20%|                                                                    | 2/10 [00:27<01:48, 13.53s/it]
Epoch 3:  83%|                                                          | 496/600 [00:14<00:03, 27.88it/s]
```

図 2: tqdm を使用した様子

  学習全体の進行度と現在学習中のエポックにおける進行度の 2 つをプログレスパーで表示している。

- クロスエントロピー誤差の値が落ちていっている様子を数字ではなく視覚的にも捉えることができるように, グラフで表示するようにした。

- 課題 2 のプログラムでは, バッチの複数枚画像に対する計算を map 関数で行っていたが, 課題 3 で行列演算で複数枚の画像を一度に処理するように実装した。これによって, 中間層ノード数 100, エポック数 10 における学習までの時間が 145 秒から 113 秒へと, およそ 1.28 倍の高速化に成功した。

- プログラム実行時に, 外部のパラメータファイルを利用するかどうか, 学習したパラメータをファイルに保存するかどうかを yes, no で入力することで決定できるように実装した。

5 問題点

- シグモイド関数内で exp 演算を行うときに overflow が発生する場合がある。ベクトルを入力とするシグモイド関数を, for ループで各要素に対して exp 演算を行うのではなく, numpy の exp 関数を利用して定義しているので, overflow への対処が難しい。