# 計算機科学実験及演習4 画像処理 レポート4

橘大佑
1029-31-6811
2019年度入学

2021/11/30

工学部情報学科計算機科学コース
1029-31-6811
橘 大佑

# レポート 4



## 課題内容

MNIST のテスト画像 1 枚を入力とし、3 層ニューラルネットワークを用いて、0 から 9 の値のうち 1 つを出力するプログラムを作成した。

## プログラムの説明

課題 3 のコードを用いて作成したパラメータを利用して、入力された整数番目の画像の推定した数字と正しい数字の 2 種類を出力するようにした。また、60000 枚の画像データについて正しい数字と学習済みのパラメータを用いて推定した数字が正しいかどうかを計算し、正答率を出力できるようにした。

- show_accuracy 関数
  画像 60000 枚のうち、推定した数字と実際の正しい数字が一致している確率を計算し、出力する。

Listing 1: show_accuracy

```
def show_accuracy():
  true_cnt = 0
  false_cnt = 0
  for i in range(60000):
    print("\r"+str(i+1),end="/60000...")
    time.sleep(0.00)
    estimated_ans = output_answer(fully_connected_layer_2(
        fully_connected_layer_1(input_layer(i))))
    real_ans = Y[i]
    if estimated_ans == real_ans:
      true_cnt += 1
    else:
      false_cnt += 1
  print("")
  print("Accuracy is ", true_cnt/(true_cnt+false_cnt))
```

## 実行結果

以下が実行したときの出力である。

```
Input number(0~9999)!!!>>> 89
True number is  4
Estimated answer is  4
Do you want to calculate accuracy?(Y/N).It will take some time.>>> Y
60000/60000...
Accuracy is  0.9915833333333334
```

Accuracy の通り、99 パーセントの確率で推定値と正しい値は一致する。また、クロスエントロピー誤差のエポックごとの値の変化の様子を以下に記述する。

```
Do you reuse parameter files?(Y/N) N
No
1
0.49116619113043347
2
0.20507840162665536
3
0.15741328272003627
...
28
0.024456560528518582
29
0.02417783506065879
30
0.021415334155113066
...
49
0.010444645169308731
50
0.0073431466799720605
60
0.0068697907403160825
```

クロスエントロピー誤差は 1 エポック目では 0.49 程度であったのが、30 エポック目では 0.021 程度まで、60 エポック目では 0.006 まで減少しているのが分かる。

### 工夫点

- 正しい入力の提案
  0 から 9999 までの整数が入力されたときみ計算を行い、0 以下や 10000 以上の整数や、そもそも整数以外の文字列などが入力された場合は再度入力を促すようにした。

- クロスエントロピー誤差の計算
  課題 2 では 2 つの配列の要素ごとに計算していた配列 $y_k$ の特徴を用いてベクトルの計算を簡単化した。その具体的な方法を下で説明する。

  $$E = \sum_{n=1}^{C} \left( -y_k \log y_k^{(2)} \right)$$

  はじめは、この計算を $C$ 回行っていたが、ベクトル $y_k$ は

  $$y_k = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$$

  などのように 10 個の要素のうち、1 つだけが 1 で、そのほかの 9 個は 0 であるため、C(10) 回の計算のうち、9 回は $0 * \log y_k^{(2)}$ の計算を行うことになるが、この値は 0 となるため、計算しなくても

# レポート 4

よい。よって、配列 $y_k$ の要素の何番目が 1 であるのかが分かれば、1 が配列 $y_k$ の index 番目であるとして、

$$\sum_{n=1}^{C}\left(-y_k \log y_k^{(2)}\right) = -\log y_k^{(2)}[index]$$

と変形することができる。1 が何番目にあるのかを計算するには 1 が配列 $y_k$ の中で最大値であるということに着目して、argmax 関数を適用すればよい。こうすることで、10 回の計算を 1 回に減らすことができた。以下がそれぞれのコードである。

Listing 2: 愚直に配列の要素を計算するコード

```
def log_fun(x):
  return math.log(x)

new_func = np.frompyfunc(log_fun, 1, 1)
z = new_func(output_l)
error -= np.dot(z, one_hot)
```

Listing 3: argmax を用いた改良コード

```
error -= math.log(output_l[np.argmax(one_hot)])
```

- クロスエントロピー誤差の描画
  試行回数とその時のクロスエントロピー誤差を出力するだけではクロスエントロピー誤差の変動を理解しにくいため、それを plot してグラフで表現した。ランダムなパラメータを利用した場合のエントロピー誤差の推移が以下である。

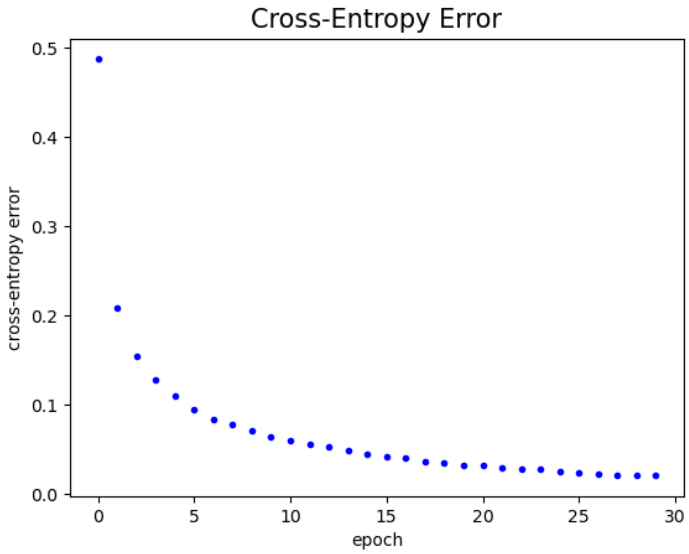


図 1: クロスエントロピー誤差の推移 (30 エポック)

# レポート 4

図 1は 30 エポックだけ計算したときのクロスエントロピー誤差をプロットしたものである。図 1から、エントロピー誤差が最初は 0.5 程度であったのが、30 回の試行後には 0.03 程度まで減少しているのが分かる。

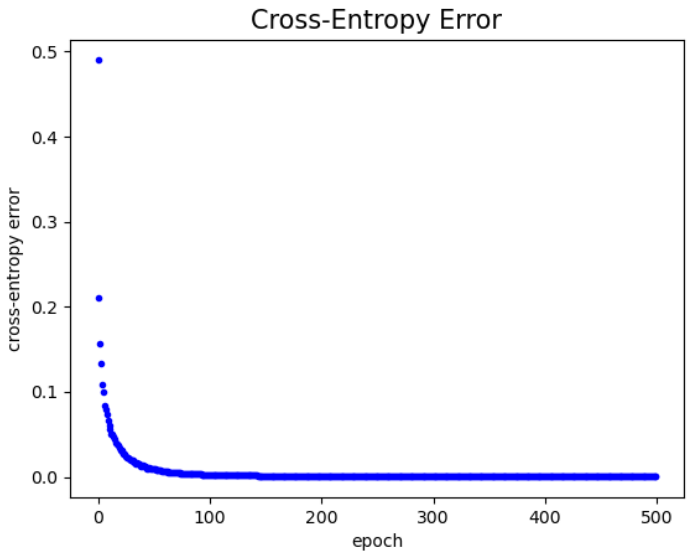


図 2: クロスエントロピー誤差の推移 (500 エポック)

図 2は暇なときに 500 エポックだけ計算したときのクロスエントロピー誤差をエポックごとにプロットしたものである。正直、200 エポック目以降はクロスエントロピー誤差の減少はほとんど見られないので必要なかったかもしれないが、いちおう載せておく。図 2から、エントロピー誤差が最初は 0.5 程度であったのが、500 回の試行後には 0.003 程度まで減少しているのが分かる。

- 実行時間の表示 1 エポックでどれほど時間がかかるかを表示させると、1 エポックで 106 秒かかっていることが分かった。

**問題点**

- 計算量
  バッチサイズ (B) だけ計算するのに for 文を用い、しかも繰り返し操作でも for 文を用いたため、for 文の入れ子構造ができてしまい、かなり学習に時間がかかった。具体的な計算量は (繰り返し回数)×(バッチサイズ)×(エポックの回数)=(教師データサイズ)×(エポック数)=60000×150=9000000 となった。

- 過学習
  上で 500 エポック計算した後にパラメーターファイルを保存して、それを用いて正答率を計算すると 100%になり、過学習していることが分かった。そこで、150 エポック程度に計算をとどめておくことで過学習を防止した。