計算機科学実験及演習 4 画像処理 レポート

工学部 情報学科 計算機科学コース
学生番号: 1029358455　　氏名: 登古紘平

2025 年 10 月 17 日

課題内容

[課題 1] のコードをベースに，ミニバッチ（＝複数枚の画像）を入力可能とするように改良し，さらにクロスエントロピー誤差を計算するプログラムを作成せよ．

- MNIST の学習画像 60,000 枚の中からランダムに B 枚をミニバッチとして取り出すこと．
- クロスエントロピー誤差の平均を標準出力に出力すること．
- ニューラルネットワークの構造，重みは課題 1 と同じでよい．
- バッチサイズ B は自由に決めて良い（100 程度がちょうどよい）．
- ミニバッチを取り出す処理はランダムに行う．
- for 文を利用してミニバッチ内の各画像に対して 1 枚ずつクロスエントロピー誤差を計算するような処理とはしないこと．ミニバッチ内の画像を多次元ベクトルとして一括で処理すること．
- 課題 1 で実装したソフトマックス関数をそのまま適用すると，上手く動作しない可能性がある．正規化をする（和を取る）範囲を意識した実装とすること．
- この時点ではランダムな予測結果を返すプログラムのため，クロスエントロピー誤差は大体 $-\log(1/10) = 2.3$ 前後になる．

1 作成したプログラムの説明

このプログラムは、クロスエントロピー誤差の平均を計算するプログラムである。課題 1 でリファクタリングを行った後のコードをベースに実装した。クロスエントロピー誤差平均の計算に必要な情報は、バッチサイズ、one-hot vector 表記の正解ラベル、および出力層における各要素の出力である。課題 1 の時点で順伝播は実装済みであるため、本課題では新たに以下の関数を実装した。

- `get_random_index`: インデックスをランダムに取得
- `get_batch_image_vector`: ベクトルを取得
- `get_batch_image_label`: 対応するラベルを取得
- `get_one_hot_label`: 正解ラベルを one-hot vector に変換
- `get_cross_entropy_error`: クロスエントロピー誤差平均を計算

課題 1 では隠れ層のユニット数を 10 個にしていたが、誤差が大きかったため 100 個に変更した。また、今回はバッチサイズ分の画像 100 枚を多次元ベクトルとして一括で計算した。

課題 1:

```
hidden_layer_input = np.dot(weight1, input_vector) + bias1
output_layer_input = np.dot(weight2, hidden_layer_output) + bias2
```

課題 2:

```
hidden_layer_input = np.dot(input_vector, weight1.T) + bias1
output_layer_input = np.dot(hidden_layer_output, weight2.T) + bias2
```

上記のコードは、順伝播を計算する関数の一部である。課題 1 のコードは、 input_vector が 1 枚の画像（1 次元ベクトル）である場合にのみ正しく機能するものであった。しかし、課題 2 ではミニバッチ処理を導入し、input_vector が 100 枚の画像をまとめた多次元ベクトル（形状: (100, 784)）に変化した。この変更に伴い、行列計算を正しく実行するため、 np.dot の引数の順序を入れ替えるとともに、重み行列を転置（ .T ）するよう修正した。これにより、入力データが「バッチサイズ × 入力層の次元数」の形状を持つ場合でも、正しく順伝播の計算が行えるようになった。

2 実行結果

```
tokokohei@DESKTOP-9F63DRB:~/image-processing/image-processing/all
/assignment2$ python3 assignment2.py
予測されたクロスエントロピー誤差は: 2.4629320247084823 です。
```

これはターミナル上で実行した結果である。以上より、クロスエントロピー誤差の平均を課題の想定通り計算できていることがわかる。

3 工夫点

今回は、画像データだけでなく、それに対応する正解ラベルを同時に取得する必要があった。そのため、画像を直接ランダムに取得するのではなく、まずインデックスをランダムに取得し、そのインデックスに基づいて画像およびラベルを取得する手法を採用した。この一連の流れを、 get_random_index 、get_batch_image_vector 、 get_batch_image_label といった関数に分割して定義することで、コードの可読性を高めた。

また、クロスエントロピー誤差を計算する際、対数関数の引数が 0 になることを防ぐため、微小な正の値 delta を導入して計算の安定性を確保した。

4 問題点

当初の実装では、 get_batch_image_vector と get_batch_image_label の両関数が、ランダムに取得したインデックス配列とバッチサイズを引数として受け取っていた。しかし、インデックス配列の長さとバッチサイズは常に一致するため、この設計は冗長であった。当初は各関数の役割を分離することで可読性の向上を意図したが、結果として非効率な実装となっていた。

5 リファクタリング

リファクタリングには、Gemini を用いた。

- プロンプト: (コード貼り付け) このコードにおいて、計算が工夫できるところは工夫し、冗長な部分は省いてください。コメントは消さないでください。
- 目的: コードの可読性を高めること、また、自分が認識していない論理エラーや冗長性を発見すること。

リファクタリング前:

```
def get_random_index(batch_size): #インデックスをランダムに取得
    test_images_arrays = np.arange(len(test_images))
    random_index = np.random.choice(test_images_arrays, size=batch_size, replace=False)
    return random_index

def get_batch_image_vector(random_index, batch_image_number):  #ベクトルを取得
    batch_images = test_images[random_index]
    return batch_images.reshape(batch_image_number, -1)

def get_batch_image_label(random_index, batch_image_number): #ラベルを取得
    batch_labels = test_labels[random_index]
    return batch_labels
```

リファクタリング後:

```
def get_random_index(batch_size): #インデックスをランダムに取得
    # np.arange の生成を省略し、直接データ数からサンプリングする
    return np.random.choice(len(test_images), size=batch_size, replace=False)

# 画像とラベルを別々に取得する冗長な関数を一つに統合
def get_batch(random_index, batch_size):
    # ベクトルとラベルをまとめて取得
    batch_images = test_images[random_index].reshape(batch_size, -1)
    batch_labels = test_labels[random_index]
    return batch_images, batch_labels
```

変更点は以下の通りである。

- **インデックス生成の簡素化**: get_random_index 関数内で、不要なインデックス配列 (np.arange) の生成を省略した。 np.random.choice は整数の上限を直接指定できるため、これを用いてコードをより簡潔かつ効率的に修正した。
- **データ取得処理の効率化**: ミニバッチを取得する際、画像とラベルを別々の関数で取得していた冗長な処理を、一つの関数 get_batch に統合した。これにより、一度の処理で画像とラベルの両方を取得できるようになった。