課題 3 レポート

羽路 悠斗

2022 年 1 月 6 日

1　課題内容

1.1　誤差逆伝播法による 3 層ニューラルネットワークの学習

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 $W^{(1)}$, $W^{(2)}$, $b^{(1)}$, $b^{(2)}$ を学習するプログラムを作成せよ。

- ネットワークの構造、バッチサイズは課題 2 と同じで良い
- 学習には MNIST の学習データ 60000 枚を用いること
- 繰り返し回数は自由に決めて良い
- 学習率は自由に決めて良い
- 各エポックごとにクロスエントロピー誤差を標準出力に出力すること
- 学習終了時にパラメータをファイルに保存する機能を用意すること
- ファイルに保存したパラメータを読み込み、再利用できる機能を用意すること

2　作成したプログラムの説明

2.1　誤差逆伝播法

2.1.1　設計方針

課題 1、2 にて各層の順伝播を関数として実装済みである。それらの戻り値を保持しておき、逆伝播を計算する。行 (axis=0) がノードを表し、列 (axis=1) がバッチサイズを表す入出力インターフェースであることに注意しておく。以下に、逆伝播の順序に従い、出力側から順に実装を示す。なお、これらは全て Model クラス中の train_batch 関数内に実装される。

2.1.2　ソフトマックス関数

ソフトマックス関数の順伝播である

ソースコード 1　ex3.py

```
1     y2 = self.softmax(a)
```

逆伝播である

ソースコード 2　ex3.py

```
1     diff_e_n_with_a = (y2 - tr_y.T)/self.batch_size # クラス数*bs
```

2.1.3　全結合層 2

二つ目の全結合層の順伝播である

ソースコード 3　ex3.py

```
1     a = self.dense_layer2(y)
```

逆伝播である

ソースコード 4　ex3.py

```
1     diff_e_n_with_y = self.w2.T @ diff_e_n_with_a # units*bs
2     diff_e_n_with_w2 = diff_e_n_with_a @ y.T # クラス数*units
3     diff_e_n_with_b2 = np.sum(diff_e_n_with_a, axis=1) # クラス数次元ベクトル
```

2.1.4　シグモイド関数

シグモイド関数の順伝播である

ソースコード 5　ex3.py

```
1     y = self.sigmoid(t)
```

逆伝播である

ソースコード 6　ex3.py

```
1     diff_e_n_with_t = diff_e_n_with_y*(1-y)*y # units*bs
```

2.1.5 全結合層 1

一つ目の全結合層の順伝播である

ソースコード 7　ex3.py

```
1    t = self.dense_layer1(x)
```

逆伝播である

ソースコード 8　ex3.py

```
1    diff_e_n_with_x = self.w1.T @ diff_e_n_with_t # 784*bs
2    diff_e_n_with_w1 = diff_e_n_with_t @ x.T # units*784
3    diff_e_n_with_b1 = np.sum(diff_e_n_with_t, axis=1) # 数次元ベクトル units
```

2.1.6 重みの更新

パラメータは学習率と、パラメータの微分に基づいて更新される。

ソースコード 9　ex3.py

```
1    self.w1 = self.w1 - lr*diff_e_n_with_w1
2    self.b1 = self.b1 - lr*diff_e_n_with_b1
3    self.w2 = self.w2 - lr*diff_e_n_with_w2
4    self.b2 = self.b2 - lr*diff_e_n_with_b2
```

2.1.7 バッチの学習

以上の処理をまとめたものが train_batch 関数である。

ソースコード 10　ex3.py

```
1    def train_batch(self, tr_x, tr_y, lr):
2      x = self.input_layer(tr_x)
3      t = self.dense_layer1(x)
4      y = self.sigmoid(t)
5      a = self.dense_layer2(y)
6      y2 = self.softmax(a)
7      e_n = self.cross_entropy(tr_y.T, y2)
8
9      # 以下微分の形を行列で示す*
10     diff_e_n_with_a = (y2 - tr_y.T)/self.batch_size # クラス数*bs
11     diff_e_n_with_y = self.w2.T @ diff_e_n_with_a # units*bs
12     diff_e_n_with_w2 = diff_e_n_with_a @ y.T # クラス数*units
13     diff_e_n_with_b2 = np.sum(diff_e_n_with_a, axis=1) # クラス数次元ベクトル
14     diff_e_n_with_t = diff_e_n_with_y*(1-y)*y # units*bs
15     diff_e_n_with_x = self.w1.T @ diff_e_n_with_t # 784*bs
16     diff_e_n_with_w1 = diff_e_n_with_t @ x.T # units*784
17     diff_e_n_with_b1 = np.sum(diff_e_n_with_t, axis=1) # 数次元ベクトル units
```

```
18
19      # 重みの更新
20      self.w1 = self.w1 - lr*diff_e_n_with_w1
21      self.b1 = self.b1 - lr*diff_e_n_with_b1
22      self.w2 = self.w2 - lr*diff_e_n_with_w2
23      self.b2 = self.b2 - lr*diff_e_n_with_b2
24
25      return e_n
```

2.2 モデルの初期化と学習ループ

2.2.1 初期化

モデルの初期化の際に、各層の重みを初期化する。

ソースコード 11　ex3.py

```
1   class Model:
2     def __init__(self) -> None:
3       # モデルのアーキテクチャを作成
4       self.units = 32
5       self.batch_size = 100
6       self.w1, self.b1 = random.normal(loc=0, scale=np.sqrt(1/784), size=784*self.
            units).reshape(self.units, 784), random.normal(loc=0, scale=np.sqrt(1/784)
            , size=self.units)
7       self.w2, self.b2 = random.normal(loc=0, scale=np.sqrt(1/self.units), size=self.
            units*10).reshape(10, self.units), random.normal(loc=0, scale=np.sqrt(1/
            self.units), size=10)
```

2.2.2 学習ループ

学習ループは、学習データ、エポック数、学習率を受け取る関数として実装する。またエポック終了時に、そのエポックのミニバッチのクロスエントロピー誤差の平均を標準出力に出力する。デフォルト値として、エポック数 10、学習率 0.01 を用いる。

ソースコード 12　ex3.py

```
1     def train(self, train_x, train_y, epochs: int = 10, lr: float = 0.01):
2       '''
3       train_x: 学習データの特徴量
4       train_y: 学習データのラベル
5       epochs: エポック数
6       lr: 学習率
7       '''
8       for i in tqdm(range(epochs)):
9         entropies = []
10        for j in range(60000//self.batch_size):
11          tr_x, tr_y = self.preprocessing(train_x, train_y)
12          entropies.append(self.train_batch(tr_x, tr_y, lr))
13        print(f'Epoch␣{i+1}␣end!␣Cross␣entroppy␣is␣{sum(entropies)/len(entropies)}.')
```

2.3 重みの保存と読み込み

2.3.1 重みの保存

model ディレクトリを作成し、重みを保存する。

ソースコード 13 ex3.py

```
1    def save_model(self, name):
2      np.savez(f'model/{name}', w1=self.w1, b1=self.b1, w2=self.w2, b2=self.b2)
```

2.3.2 重みの読み込み

model ディレクトリから、重みを読み込む。各パラメータについて、保存時の名前を明示する。

ソースコード 14 ex3.py

```
1    def load_model(self, name):
2      model = np.load(f'model/{name}.npz')
3      self.w1, self.b1, self.w2, self.b2 = model['w1'], model['b1'], model['w2'], model
           ['b2']
```

3 実行結果

実行部分は課題 2 と同様である

ソースコード 15　ex3.py

```
    if __name__ == '__main__':
      train_x = mnist.download_and_parse_mnist_file('train-images-idx3-ubyte.gz')
      train_y = mnist.download_and_parse_mnist_file("train-labels-idx1-ubyte.gz")
      model = Model()
      model.train(train_x, train_y)
```

実行結果

```
  0%|                    | 0/10 [00:00<?, ?it/s]
Epoch 1 end! Cross entroppy is 1.2626488958020
09.
 10%|          | 1/10 [00:01<00:13,  1.52s/it]
/Users/yuto/Documents/university/32_exercise4_
im/ex3.py:36: RuntimeWarning: overflow encount
ered in exp
  return 1/(1+np.exp(-input_vec))
Epoch 2 end! Cross entroppy is 0.7387793511273
026.
 20%|          | 2/10 [00:02<00:11,  1.39s/it]
Epoch 3 end! Cross entroppy is 0.5738505853220
85.
 30%|          | 3/10 [00:04<00:09,  1.43s/it]
Epoch 4 end! Cross entroppy is 0.4952071535487
975.
 40%|          | 4/10 [00:05<00:08,  1.35s/it]
Epoch 5 end! Cross entroppy is 0.4320013057128
824.
 50%|          | 5/10 [00:06<00:06,  1.33s/it]
Epoch 6 end! Cross entroppy is 0.4046619390488
334.
 60%|          | 6/10 [00:08<00:05,  1.34s/it]
Epoch 7 end! Cross entroppy is 0.3797327529576
6973.
 70%|          | 7/10 [00:09<00:04,  1.34s/it]
Epoch 8 end! Cross entroppy is 0.3648856010381
47.
 80%|          | 8/10 [00:10<00:02,  1.36s/it]
Epoch 9 end! Cross entroppy is 0.3421980485568
8857.
 90%|          | 9/10 [00:12<00:01,  1.33s/it]
Epoch 10 end! Cross entroppy is 0.334565611355
4381.
```

クロスエントロピー誤差が減少していっている様子が観察できる。まだ収束しているとは言い難いのでエポック数を増やせばさらに損失を減らせると考えられる。

4　工夫点

各層の入力と出力のインターフェースを統一していたことで、逆伝播の実装も、テキストにあった式を愚直に numpy で表現していくだけで済んだ。今回は学習ループが必要となったので、Model クラスを作成し、外部から model.train を呼ぶだけで良いようにした。このように、できる限り処理を切り分けるようにした。また今回も課題 1、2 と同様に、各層の演算に for 文を使わずに行列演算を駆使したことで、高速化できた。

5　問題点

逆伝播を関数化していないことや、層のパラメータを Model クラスが持っていることが、層の拡張性や、モデル構造の可変性を下げている。この問題は発展課題に取り組むにつれて顕在化した問題である。後の実装では、大幅に実装方針を変えて、これらの問題に対処した。