計算機科学実験及演習4「画像認識」 課題3

工学部 情報学科 計算機科学コース 3回生
山田瑛平 学籍番号:1029282731

平成30年11月2日

課題内容

以下のようなプログラムを作成する.

- 3 層ニューラルネットの重み W_1, W_2, b_1, b_2 を学習する.
- 学習した重みをファイルに保存する.
- ファイルから重みを読み込んで再利用できる.

課題内容としては以上が必要十分条件だが, それに加えて画像 60000 枚を学習群 50000 枚と試験群 10000 枚に分けた上で, 学習群によりパラメータを定め, 残り 10000 枚によりその精度を測定し, 出力するようなプログラムを作成した.

プログラムの説明

ハイパーパラメータ

先述の通り, 学習に用いる画像数は 50000 である. 中間層の数は 500, バッチサイズは 100 で, 学習率は 0.01 に設定されている. 当プログラムでは暫定的に 1 エポックだけループを回すような初期設定がされている.

パラメータのインポート

このプログラムは同じディレクトリに位置するファイル $'ex3\_w1.npy'$, $'ex3\_w2.npy'$, $'ex3\_b1.npy'$, $'ex3\_b2.npy'$ からパラメータを読み込む. しかし, ただファイルをロードするような実装を行っただけでは, ファイルが存在しない場合にエラーを出力してしまう. そこで, プログラムを以下のような仕様に変更した.

- 実行時に標準入力を受け付ける.
- その際の入力が 1 ならパラメータを読み込まずに, ガウス分布による乱数をパラメータの初期値として利用する.
- それ以外の文字列が入力されれば, 当該ファイルに格納されたパラメータをそのまま重みとして用いる.

同一ディレクトリにファイルの存在しない初期状態から学習を行う場合, 1 を入力すれば, 学習後 (すなわちプログラムの実行後) に新たにファイルが作成される.

学習フェーズ

課題 2 のプログラムとの大きな相違点は, (言うまでもなく) 学習フェーズである. 実装方針の概要については教科書に記載があるので, ここでは実際の実装とバックプロパゲーションの式を照らし合わせていく.

フェーズ 4

フェーズ 4 では, $\frac{\partial E_n}{\partial a_k}$ を計算する. $\frac{\partial E_n}{\partial a_k}$ は,

$$\frac{\partial E_n}{\partial a_k} = \frac{y_k^{(2)} - y_k}{B} \tag{1}$$

によって与えられる. 実装は以下の通りである.

```
en_ak = np.array((sofy2.T - k) / BATCH_SIZE)
```

フェーズ 5

フェーズ 5 では, $\frac{\partial E_n}{\partial X_2}$, $\frac{\partial E_n}{\partial W_2}$, $\frac{\partial E_n}{\partial b_2}$ を計算する. X_2 は中間層が出力したシグモイド関数適用後の値 $\sigma(W_1 x + b_1)$ の行列である. 各微分値は,

$$\frac{\partial E_n}{\partial X_2} = W_2^T \frac{E_n}{a_k} \tag{2}$$

$$\frac{\partial E_n}{\partial W_2} = \frac{E_n}{a_k} X_2^T \tag{3}$$

$$\frac{\partial E_n}{\partial b_2} = rowSum\left(\frac{E_n}{a_k}\right) \tag{4}$$

によって与えられる. 実装は以下の通りである.

```
en_x_2 = np.dot(w2.T, en_ak.T)
en_w_2 = np.dot(en_ak.T, y1.T)
en_b_2 = rowsum(en_ak.T)
```

ただし, *rowsum* 関数は以下のように定義される.

```
def rowsum(a):
    comsum = np.sum(a, axis=1)
    return np.c_[comsum]
```

フェーズ 6

フェーズ 6 では, $\frac{\partial E_n}{\partial Y}$ を計算する. Y は入力層が出力した値 $W_1 x + b_1$ の行列である. $\frac{\partial E_n}{\partial Y}$ は,

$$\frac{\partial E_n}{\partial Y} = \frac{E_n}{X_2} \circ \left(1 - \sigma\left(\frac{E_n}{X_2}\right)\right) \sigma\left(\frac{E_n}{X_2}\right) \tag{5}$$

によって与えられる. $\circ$ はアマダール積を表す. 実装は以下の通りである.

```
en_y_1 = en_x_2 * sigdarr(en_x_2)
```

ただし, *sigdarr* 関数はシグモイド関数の微分を行列の全要素に適用する関数であり, 以下のように定義される.

```
def sigd(t):
    return (1 - sig(t)) * sig(t)

def sigdarr(a):
    return np.vectorize(sigd)(a)
```

フェーズ 7

フェーズ 7 では, $\frac{\partial E_n}{\partial X}$, $\frac{\partial E_n}{\partial W_1}$, $\frac{\partial E_n}{\partial b_1}$ を計算する. X はニューラルネット全体への入力となる $784 \times B$ 行列である. 各微分値は,

$$\frac{\partial E_n}{\partial X} = W_1^T \frac{E_n}{Y} \tag{6}$$

$$\frac{\partial E_n}{\partial W_1} = \frac{E_n}{Y} X_1^T \tag{7}$$

$$\frac{\partial E_n}{\partial b_1} = rowSum\left(\frac{E_n}{Y}\right) \tag{8}$$

によって与えられる. 実装は以下の通りである.

```
en_x_1 = np.dot(w1.T, en_y_1)
en_w_1 = np.dot(en_y_1, x_input)
en_b_1 = rowsum(en_y_1)
```

フェーズ 8

フェーズ 8 では, 学習率 η を用いて, 重みの更新を行う. 微分値に学習率を掛けた値を元の重みから減じることで更新が完了する. 実装は以下の通りである.

```
w1 = w1 - LEARNING_RATE * en_w_1
w2 = w2 - LEARNING_RATE * en_w_2
b1 = b1 - LEARNING_RATE * en_b_1
b2 = b2 - LEARNING_RATE * en_b_2
```

精度の計測

試験群 10000 枚の画像すべてについて走査する. ニューラルネットが正しい答えを出力すれば正答数に 1 を加算, 間違えた答えを出力すれば誤答数に 1 を加算し, 最終的な正答率を出力する.

実行結果

1 エポックで平均クロスエントロピー誤差は 0.2 から 0.5 にかけての値を不規則に出力するようになり, 精度は概ね 91%前後の値を弾き出す. また, エポックの回数を増やしても, 平均クロスエントロピー誤差の値や精度に向上は見られなかった.

工夫点および問題点

学習は正しく行われている. 工夫できる点があるとするならばハイパーパラメータの選択である. 当プログラムの中間層の数の初期値は 500 であるが, この値には選択の余地がある. 中間層の数を増やせばその分入力画像の情報をより吸収できるが, 計算量の増加により学習にかかる時間が大きくなる.
そこで, 中間層の数を上下させ, どのような精度を得られるかを実験した.

- 中間層 =100 のとき, 精度はおよそ 89.5% であった.
- 中間層 =200 のとき, 精度はおよそ 90.1% であった.
- 中間層 =300 のとき, 精度はおよそ 90.8% であった.
- 中間層 =400 のとき, 精度はおよそ 91.2% であった.
- 中間層 =600 のとき, 精度はおよそ 91.3% であった.
- 中間層 =700 のとき, 精度はおよそ 91.4% であった.

このように, 中間層の数の増加は精度の向上に繋がる. しかし中間層の数 =700 程度で精度は収束し, これ以上中間層の数を増やしても見返りが少ないうえ, 学習に必要な時間が無意味に増大するだけになる. このため今回は中間層の数を 500 に固定している.