発展課題レポート

情報学科 計算機コース 1029275871 芦田聖太

提出日　18/1/26

発展課題

- コンテスト参加
- ReLU 関数の導入と adam への変更

1　コンテスト参加

1.1　設計方針

- コンテストデータの読み込み
- 結果のファイルの書き込み

1.2　実装とプログラムの説明

1.2.1　コンテストデータの読み込み

テキスト通りに pickle を使って、コンテストデータを読み込んだ。

contest.py

```
with open("/Users/omushota/ex4-image/le4MNIST_X.dump","rb") as f:
    X = pickle.load(f, encoding='bytes')
    X1 = X.reshape((X.shape[0], 28, 28))
    X = X.reshape((X.shape[0], 28 * 28))
    print(X.shape[1])
```

1.2.2　結果のファイルの書き込み

評価結果列 indexmax を savetxt を使い kaitou_sgd に保存した。

contest.py

```
np.savetxt("kaitou_sgd.txt", indexmax, fmt="%d")
```

1.3　考察

adam で改良したものより最初の設計である sgd の方が良い精度で認識ができた。しかし、学習速度は adam の方が早かった。学習速度と正答率は完全に比例するわけではないようである。畳み込みには手をだすことができなかったが、畳み込みするとどの程度認識率があがるのかも調べてみたい。

1.4　contest.py 全文

```
import pickle
import sigmoid
import softmax
import numpy as np
import matplotlib.pyplot as plt
from pylab import cm

# python 3 系の場合は import pickle としてください.
with open("/Users/omushota/ex4-image/le4MNIST_X.dump","rb") as f:
    X = pickle.load(f, encoding='bytes')
```

```
    X1 = X.reshape((X.shape[0], 28, 28))
    X = X.reshape((X.shape[0], 28 * 28))
    print(X.shape[1])

weightfile = np.load('parameters2.npz')

line = X.shape[0]
row = X.shape[1]
batch = 10000
loop = int(len(X) / batch)

counter = 0
for n in range(loop):
    print(str(n) + "回目")
    if ((n + 1) * batch) % 10000 != 0:
        learn = np.reshape(X[(n * batch) % 10000: ((n + 1) * batch) % 10000:], (batch,
            row)).T
        # print(answer)
    else:
        learn = np.reshape(X[(n * batch) % 10000: 10000:], (batch, row)).T

    # 中間層##################################
    # 定数
    middle = 300

    # 重み1
    weight1 = weightfile['w1']
    b1 = weightfile['b1']

    # 中間層への入力
    midinput = weight1.dot(learn) + b1

    # シグモイド
    midout = sigmoid.sigmoid(midinput)

    # 出力層##################################
    # 定数
    end = 10

    # 重み2
    weight2 = weightfile['w2']
    b2 = weightfile['b2']

    # 出力層への入力
    fininput = weight2.dot(midout) + b2

    # ソフトマックス
    finout = softmax.softmax(fininput)
    indexmax = finout.argmax(axis=0)
    print("indexmax")
    print(indexmax)

np.savetxt("kaitou_sgd.txt", indexmax, fmt="%d")
```

2 ReLU 関数と adam

2.1 設計方針

- ReLU 関数と ReLU の逆伝播の設計
- ニューラル内への導入
- 重みの修正の仕方の変更（adam）

2.2 実装とプログラムの説明

2.2.1 ReLU 関数と ReLU の逆伝播の設計

ReLU 関数の定義より、入力が 0 より大きい時は入力をそのまま出力し、その他の場合は 0 を返す関数を作成した。また、その逆伝播では入力が 0 より大きい時は 1, その他の場合は 0 を返す。行列が入力された場合は各要素ごとに計算されるように vectorize している。

funcs.py

```
@np.vectorize
def ReLU(x):
    if x > 0:
        return x
    else:
        return 0.0

@np.vectorize
def dif_ReLU(x):
    if x > 0:
        return 1
    else:
        return 0
```

2.2.2 ニューラル内に導入

もともと sigmoid で計算していた部分を ReLU に変更するだけである。逆伝播の部分も、dif_ReLU を用いるだけである。

main2.py

```
# 中間層
midout = funcs.ReLU(midin)

# 逆伝播2
aen_ay1 = aen_ax2 * funcs.dif_ReLU(midin)
```

2.2.3 重みの修正方法の変更 (adam)

$$t = t + 1 \tag{1}$$

$$m = \beta_1 m + (1 - \beta_1)\frac{\partial E_n}{\partial W} \tag{2}$$

$$v = \beta_2 v + (1 - \beta_2)\frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \tag{3}$$

$$\hat{m} = \frac{m}{1 - \beta_1^t} \tag{4}$$

$$\hat{v} = \frac{v}{1 - \beta_2^t} \tag{5}$$

$$W_t = W_{(t-1)} - \frac{\alpha \hat{m}}{\sqrt{\hat{v}} + \epsilon} \tag{6}$$

以上の 5 つの式の規則のもと重みの更新を行う。w1, w2, b1, b2 のそれぞれについて同じ手順で更新を行う。ハイパーパラメータの初期値はプログラムの最初に書かれた初期値の部分に表されている値を用いた。t で表している部分はプログラム中では n を用いて表されている。

```
#初期値
alpha = 0.001
beta1 = 0.9
```

```
beta2 = 0.999
e = 0.00000001

    # 重み修正
    # adam
    m_w1 = beta1 * m_w1 + (1 - beta1) * aen_aw1
    v_w1 = beta2 * v_w1 + (1 - beta2) * aen_aw1 * aen_aw1
    m_w1_dash = m_w1 / (1 - beta1 ** (n + 1))
    v_w1_dash = v_w1 / (1 - beta2 ** (n + 1))
    weight1 -= alpha * m_w1_dash / (np.sqrt(v_w1_dash) + e)

    m_b1 = beta1 * m_b1 + (1 - beta1) * aen_ab1
    v_b1 = beta2 * v_b1 + (1 - beta2) * aen_ab1 * aen_ab1
    m_b1_dash = m_b1 / (1 - beta1 ** (n + 1))
    v_b1_dash = v_b1 / (1 - beta2 ** (n + 1))
    b1 -= alpha * m_b1_dash / (np.sqrt(v_b1_dash) + e)

    m_w2 = beta1 * m_w2 + (1 - beta1) * aen_aw2
    v_w2 = beta2 * v_w2 + (1 - beta2) * aen_aw2 * aen_aw2
    m_w2_dash = m_w2 / (1 - beta1 ** (n + 1))
    v_w2_dash = v_w2 / (1 - beta2 ** (n + 1))
    weight2 -= alpha * m_w2_dash / (np.sqrt(v_w2_dash) + e)

    m_b2 = beta1 * m_b2 + (1 - beta1) * aen_ab2
    v_b2 = beta2 * v_b2 + (1 - beta2) * aen_ab2 * aen_ab2
    m_b2_dash = m_b2 / (1 - beta1 ** (n + 1))
    v_b2_dash = v_b2 / (1 - beta2 ** (n + 1))
    b2 -= alpha * m_b2_dash / (np.sqrt(v_b2_dash) + e)
```

2.3 実行結果

学習を行い、check.py で正答率を調べたところ。以下のような実行結果が得られた。

```
# 正答率
96.89999999999999
```

2.4 考察

2.4.1 工夫点

ReLU 関数を vectorize で定義することで、main のプログラムを見やすくすることができた。さらに、ReLU 関数では学習速度が今までの倍程度になった。また、定数をうまく定義することで adam の部分を簡単に処理することができた。

2.4.2 問題点

正答率が以前より下がってしまっている。画像の認識にはある程度の限界があるようではあるが、他の発展課題をしてみるともう少し正答率があがるのかもしれない。