# 課題1レポート

芦田聖太

提出日　17/12/21

# 課題 1

# 1　3 層のニューラルネットワークの構築

MNIST の画像 1 枚を入力とし，3 層ニューラルネットワークを用いて，0〜9 の値のうち 1 つを出力するプログラムを作成する。

### 仕様

- キーボードから 0〜9999 の整数を入力 i として受け取り，0〜9 の整数を標準出力に出力すること。
- MNIST のテストデータ 10000 枚の画像のうち i 番目の画像を入力画像として用いる。（ただし、MNIST の画像サイズ（28 × 28）, 画像枚数（10000 枚）, クラス数（C = 10）は既知とする。）
- 中間層のノード数 M は自由に決めて良い。
- 重み W(1), W(2), b(1), b(2) については乱数で決定すること。ここでは，手前の層のノード数を N として 1/N を分散とする平均 0 の正規分布で与えることとする。実行する度に同じ結果を出力するよう乱数のシードを固定すること 。

# 2　設計方針

仕様を満たす 3 層のニューラルネットワークを構築するのに必要なものを以下にあげる。これらを設計し、組み合わせることで 3 層のニューラルネットワークを構成する。

- キーボードからの入力処理と画像の取り込み
- 中間層への入力と出力層の入力の計算
- シグモイド関数
- ソフトマックス関数

# 3　実装とプログラムの説明

## 3.1　キーボードからの入力処理と画像の取り込み

X, Y にテストデータを取り込み、X は (10000, 28, 28) の 3 次元の配列に整える。Y は正解のラベルの集合なので長さ 10000 の配列とする。
キーボードの入力に関しては、while ループの中で入力を strnum に格納し、数字が 0 9999 の場合は break する。それ以外の場合はループして 0 9999 の数字が入力されるまで待つ。break 後、X の入力された番号の画像データ（28 , 28）を indata に格納しさらに（784, 1）の配列へと変換する。

main.py

```
# 入力
# 画像取り込み
mndata = MNIST("/Users/omushota/ex4-image/le4nn")
X, Y = mndata.load_testing()
X = np.array(X)
X = X.reshape((X.shape[0], 28, 28))
Y = np.array(Y)


# キーボード入力待ち
while True:
    strnum = input("input␣number␣:␣")
    num = int(strnum)
    if (num < 0) or (num > 9999):
        print("Please␣type␣0␣~␣9999")
    else:
        break

indata = X[num]
line = X.shape[0]
row = X.shape[1]
indata = np.reshape(indata, (row * row, 1))
```

### 3.2 中間層への入力と出力層への入力

中間層への入力と出力層への入力に関しては同じ処理を行うので、中間層への入力についてのみについて述べる。課題 2 で同じ処理を複数のデータについて行うことを考慮して、中間層への入力を計算する関数を構成した。関数では、入力データ, 中間層の数, 入力層の数, 平均, 分散, シード値を入力とする。まず、np.random.seed(seed) で乱数のシード値を設定することで、実行するたびに同じ乱数が生成されるようにする。次に、row * middle の長さの乱数配列を発生させ、(middle, row) の 2 次元配列にし weight に格納する。また、同様に middle の長さの乱数配列を発生させ、(middle, 1) の 2 次元配列にし b に格納する。最後に、weight と入力データの積に b を足したものを返す。この関数を main.py で用いて中間層への入力とした。

layer.py

```
import numpy as np


def mid(indata, middle, row, average, variance, seed):
    np.random.seed(seed)
    weight = np.random.normal(average, variance, row * middle)
    weight = np.reshape(weight, (middle, row))
    b = np.random.normal(average, variance, middle)
    b = np.reshape(b, (middle, 1))
    return weight.dot(indata) + b

def endend(midout, end, middle, average, variance, seed):
    np.random.seed(seed)
    weight1 = np.random.normal(0, variance, middle * end)
    weight1 = np.reshape(weight1, (end, middle))
    b1 = np.random.normal(average, variance, end)
    b1 = np.reshape(b1, (end, 1))
    return weight1.dot(midout) + b1
```

main.py

```
# 中間層
middle = 4
average = 0
variance = math.sqrt(1/line)
```

```
seed = 100
midinput = mid(indata, middle, row * row, average, variance, seed)

~~~~~~

# 出力層
end = 10
average1 = 0
variance1 = math.sqrt(1/middle)
fininput = endend(midout, end, middle, average1, variance1, seed)
```

### 3.3 シグモイド関数

基本的に返り値は計算式のままであるが、入力によって少しだけ出力を変えている。オーバーフローの処理のために、入力が 34.538776394910684 より大きいときは 1.0 - 1e-15 を出力する。また、入力が-34.538776394910684 より小さいときは 1e-15 を出力する。そしてそれ以外の時は、1.0 / (1.0 + np.exp(-x)) を出力するようにした。

sigmoid.py

```
import numpy as np

@np.vectorize

def sigmoid(x):
    sigmoid_range = 34.538776394910684
    if x <= -sigmoid_range:
        return 1e-15
    if x >= sigmoid_range:
        return 1.0 - 1e-15
    return 1.0 / (1.0 + np.exp(-x))
```

### 3.4 ソフトマックス関数

c に np.max で配列中の 1 番大きな値を格納する。a-c で a 中の各要素からそれぞれ c を引いた配列を作り、np.exp(a-c) で各要素がネイピア数の指数となった配列を exp_a に格納する。最後に exp_a の総和を計算し sum_exp_a として exp_a / sum_exp_a を返す。

softmax.py

```
import numpy as np


def softmax(a):
    # 一番大きい値を取得
    c = np.max(a)
    # 各要素から一番大きな値を引く（オーバーフロー対策）
    exp_a = np.exp(a - c)
    sum_exp_a = np.sum(exp_a)
    # 要素の値全体の要素の合計/
    y = exp_a / sum_exp_a

    return y
```

### 3.5 全体

今まで説明したものを合わせたものを、以下に載せる。最後の部分が未説明なのでその部分だけ説明を加える。最後の部分ではソフトマックスの出力の配列の最大値のインデックスを取り出し、出力している。

main.py

```
import sigmoid
import softmax
import  CrossEntropy
import math
import numpy as np
from mnist import MNIST
from layer import mid, endend


# 入力層
# 画像取り込み
mndata = MNIST("/Users/omushota/ex4-image/le4nn")
X, Y = mndata.load_testing()
X = np.array(X)
X = X.reshape((X.shape[0], 28, 28))
Y = np.array(Y)


# キーボード入力待ち
while True:
    strnum = input("input␣number␣:␣")
    num = int(strnum)
    if (num < 0) or (num > 9999):
        print("Please␣type␣0␣~␣9999")
    else:
        # print("break")
        break

indata = X[num]
line = X.shape[0]
row = X.shape[1]
indata = np.reshape(indata, (row * row, 1))


# 中間層
middle = 4
average = 0
variance = math.sqrt(1/line)
seed = 100
midinput = mid(indata, middle, row, average, variance, seed)

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


# 出力層
end = 10
average1 = 0
variance1 = math.sqrt(1/middle)
fininput = endend(midout, end, middle, average1, variance1, seed)

# ソフトマックス
finout = softmax.softmax(fininput)
indexmax = np.where(finout == finout.max())[0][0]
print(indexmax)
```

# 4　実行結果

以下に 2 つの実行結果を示す。input の値が変わることで出力結果も変わることがわかる。また、0 9999 の数字でしか次に進まないことが確認できた。

```
input number : 10
6

Process finished with exit code 0
```

```
input number : 999999999
Please type 0 ~ 9999
input number : -13
Please type 0 ~ 9999
input number : 12
4

Process finished with exit code 0
```

# 5 考察

## 5.1 工夫点

入力に関しては、0 9999 までの数字が入らない限り待ち続けるようにした部分が工夫した点と言える。また、シグモイド関数に関してもオーバーフロー対策として入力によって値を変えた部分が自分なりに工夫した部分である。さらに、中間層への入力と出力層への入力を関数を生成して計算した部分も工夫した点である。そうすることで、課題 2 のミニバッチ処理がスムーズになることが予想される。

## 5.2 問題点

課題 3 の逆伝播の部分で重みを更新していかなければならないが、この仕様のままだと重みが関数に隠れているので、重みの更新が今の所問題点として考えられる。