

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

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、 $1.0 / (1.0 + \text{np.exp}(-x))$ 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

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

```