２０２０年度特別演習　山下担当分　課題

1AS 岩崎悠紀

2020 年 6 月 15 日

1　Chapter 1

1.1　Exercise 1.6

■ **課題内容**　Consider a following neural network which have 4 neurons in the hidden layer and 3 neurons in the output layer. Calculate outputs where the inputs x, weights w, u and thresholds h, g are given as follows by hand calculation.

■ **解答**　手計算で x, w, h, u, g を使って o(output) を出せ，という問題だった．各変数を以下のように定義する．

$$\boldsymbol{x} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} 3 & 2 & 2 & 0 \\ 4 & 1 & 5 & 2 \\ 1 & 0 & 1 & 4 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \boldsymbol{h} = \begin{bmatrix} 2 \\ 6 \\ 1 \\ 0 \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} 4 & 1 & 4 & 3 \\ 2 & 3 & 0 & 1 \\ 0 & 1 & 2 & 4 \\ 3 & 1 & 1 & 2 \end{bmatrix} \quad \boldsymbol{g} = \begin{bmatrix} 1 \\ 4 \\ 3 \\ 5 \end{bmatrix}$$

まずは一層目の $\boldsymbol{x} \cdot \boldsymbol{w}$ を計算すると，

$$\boldsymbol{w} \cdot \boldsymbol{x} = \begin{bmatrix} 5 & 5 & 3 \\ 9 & 5 & 6 \\ 2 & 1 & 5 \\ 0 & 1 & 1 \end{bmatrix}$$

となり，そこに h より大きい要素を 1，それ以外を 0 にすると以下の行列が求められる．

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

同様に，u, g を用いて o(output) は以下のように導くことができる．

$$\boldsymbol{u} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 9 & 7 & 11 \\ 5 & 3 & 3 \\ 3 & 4 & 6 \\ 5 & 5 & 6 \end{bmatrix}$$

$$\boldsymbol{o} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad (\boldsymbol{g} \text{ をフィルタとして適用})$$

■ **考察**　課題 1.6 ではニューラルネットワークの計算を手でやってみて，実際の処理のイメージを掴むことができた．また，複数個のデータを行列にまとめて計算することで効率的に結果を求めることができるということも理解できた．

2 Chapter 2

2.1 Exercise 2.3

■ **課題内容**　Based on the XOR function created in exercise 2.2, rewrite the neural network with bias term and step function (i.e., using a class “Affine.m” and “Step.m”) and make sure that the output does not change.

■ **解答**　課題内容は，前課題のソースコードを Affine と Step クラスを使うように修正せよ．また出力が変わっていないか確認する．修正したソースコードとその出力を以下に示す．

ソースコード 1　Exercise 2.3

```
import numpy as np

class Affine:
    def __init__(self, w, b):
        self.w = w
        self.b = b

    def forward(self, x):
        p = np.dot(self.w, x) + self.b
        return p

class Step:
    def forward(self, x):
        y = x > 0
        return y.astype(np.int)

x = np.array([[0,0,1,1],
              [0,1,0,1]])
w = np.array([[1, 1],
              [-1, -1]])
b = np.array([[0],
              [2]])
u = np.array([1, 1])
c = np.array([-1])

layer1 = Affine(w,b)
layer2 = Step()
layer3 = Affine(u,c)
layer4 = Step()

```

```
31 p = layer1.forward(x)
32 y = layer2.forward(p)
33 q = layer3.forward(y)
34 z = layer4.forward(q)
35 print(y)
36 print(z)
```

出力結果

```
[[0 1 1 1]
 [1 1 1 0]]
[0 1 1 0]
```

■ **考察**　本課題ではニューラルネットワークのプログラムを，Affine・Step クラスを使うように修正した．また，修正したプログラムの出力結果を前課題の出力結果と比較した．その結果出力は変えずにプログラムのみを修正することができた．

2.2 Exercise 2.4

■ **課題内容**　Implement Sigmoid.m as follows. Then, execute XOR function with sigmoid function and display the output values.

■ **解答**　課題内容はシグモイド関数のクラスを実装し，XOR の出力を表示するというものだった．ソースコードの一部とその出力結果を以下に示す．

ソースコード 2　Exercise 2.4

```
1  class Sigmoid:
2      def forward(self, x):
3          return 1 / (1 + np.exp(-x))
4  x = np.array([[0,0,1,1],
5                [0,1,0,1]])
6  w = np.array([[1, 1],
7                [-1, -1]])
8  b = np.array([[0],
9                [2]])
10 u = np.array([1, 1])
11 c = np.array([-1])
12
13 layer1 = Affine(w,b)
14 layer2 = Sigmoid()
15 layer3 = Affine(u,c)
16 layer4 = Sigmoid()
17
18 p = layer1.forward(x)
19 y = layer2.forward(p)
20 q = layer3.forward(y)
```

```
z = layer4.forward(q)
print(y)
print(z)
```

出力結果

```
[[0.5        0.73105858 0.73105858 0.88079708]
 [0.88079708 0.73105858 0.73105858 0.5       ]]
[0.59406533 0.6135163  0.6135163  0.59406533]
```

■ **考察**　本課題ではシグモイド関数のクラスを実装し，出力を表示した．シグモイド関数はステップ関数のように出力が 0 と 1 の 2 値ではなく，0〜1 の間で連続した出力が得られる．そのため上記のような出力になったと考えられる．また，後々使用することになると思うが，ステップ関数は離散なため微分できないがシグモイド関数は連続なので微分可能であるため勾配の伝搬ができる．

2.3 Exercise 2.5

■ **課題内容**　Displaying output graph with formal neuron. If we use a step function, what kind of graph is outputted?

■ **解答**　課題内容はステップ関数を用いたときの値を可視化するというものだった．ソースコードの一部とその出力結果を以下に示す．

ソースコード 3　Exercise 2.5

```
x = np.array([[ 0, 0, 1, 1],
              [ 0, 1, 0, 1]])
w = np.array([[ 2.1, 2.0],
              [-2.0,-2.0]])
b = np.array([[-1],
              [ 3]])
u = np.array([[ 2, 2]])
c = np.array([-3])

layer1 = Affine(w, b)
layer2 = Step()
layer3 = Affine(u, c)
layer4 = Step()

p = layer1.forward(x)
y = layer2.forward(p)
q = layer3.forward(y)
z = layer4.forward(q)

X, Y = np.meshgrid(np.arange(0, 1, 0.01), np.arange(0, 1, 0.01))
x = np.vstack((X.reshape(1, 10000), Y.reshape(1, 10000)))

```

```
p = layer1.forward(x)
y = layer2.forward(p)
q = layer3.forward(y)
z = layer4.forward(q)
Z = z.reshape((100, 100))

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z)
plt.show()
```

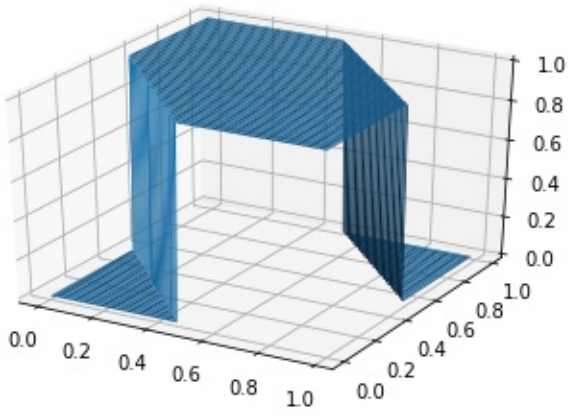

図 1　Exercise2.5 の出力結果

■ 考察　本課題では，ステップ関数を用いたニューラルネットワークの出力をグラフ化して表示した．その事により weight の値によってどうやって出力が変化しているのか視覚化されてわかりやすくなった．

Exercise 2.6

■ 課題内容　If we slightly change the value of weights or biases in exercise2_4.m, please check how the output value changes.

■ 解答　もし weights と biases の値を少し変えたら出力はどうなるのか確認せよという課題だった．ソースコードの一部とその出力結果を以下に示す．

ソースコード 4　Exercise 2.6

```
x = np.array([[0,0,1,1],
              [0,1,0,1]])
w = np.array([[1.5, 0.5],
              [-0.5, -1.1]])
b = np.array([[0],
              [1]])
u = np.array([0.5, 1.5])
```

```
c = np.array([-2])

layer1 = Affine(w,b)
layer2 = Sigmoid()
layer3 = Affine(u,c)
layer4 = Sigmoid()

p = layer1.forward(x)
y = layer2.forward(p)
q = layer3.forward(y)
z = layer4.forward(q)
print(y)
print(z)
```

出力結果

```
[[0.5        0.62245933 0.81757448 0.88079708]
 [0.73105858 0.47502081 0.62245933 0.35434369]]
[0.34222103 0.27363866 0.34129608 0.26345536]
```

■ **考察**　本課題では，weights や biases を少し変えると出力も少しだけ変わるということが分かった．これにより，ニューラルネットの保持する状態（weight や bias）を少しづつ変化させると出力も少しづつ変えることができるということが分かった．

3 Chapter 3

3.1 Exercise 3.4

■ **課題内容**　Check the values of output y, layer1.weights and layer1.bias after learning in example3_2.m.

■ **解答**　課題は学習のプログラムを実行した後のレイヤ１の weight と bias を確認せよという課題だった．ソースコードの一部とその出力結果を以下に示す．

ソースコード 5　Exercise 3.4

```
class Affine:
    def __init__(self, w, b):
        self.w = w
        self.b = b
        self.dw = None
        self.db = None
        self.input = None

    def forward(self, x):
        self.input = x
        y = np.dot(self.w, x) + self.b
```

```
12          return y
13
14      def backward(self, dx):
15          self.dw = np.dot(dx ,self.input.T)
16          self.db = np.sum(dx, axis=1, keepdims=True)
17          return np.dot(self.w.T, dx)
18
19      def update(self, lr=0.1):
20          self.w -= self.dw * lr
21          self.b -= self.db * lr
22
23  class Sigmoid:
24      def __init__(self):
25          self.output = None
26
27      def forward(self, x):
28          y = 1 / (1 + np.exp(-x))
29          self.output = y
30          return y
31
32      def backward(self, dx):
33          return dx * self.output * (1.0 - self.output)
34
35
36  x = np.array([[0, 0, 1, 1],
37                [0, 1, 0, 1]])
38  t = np.array([0, 0, 0, 1])
39
40  input_dim = 2
41  hidden_dim = 2
42  output_dim = 1
43
44  w = 2.0 * np.random.rand(1, 2) - 1.0
45  b = 2.0 * np.random.rand(1, 1) - 1.0
46
47  layer1 = Affine(w, b)
48  layer2 = Sigmoid()
49  layer3 = MSE()
50
51  epoch = 1000
52  for i in range(epoch):
53      p = layer1.forward(x)
54      y = layer2.forward(p)
55      loss = layer3.forward(y, t)
56
57      dy = layer3.backward()
58      dp = layer2.backward(dy)
```

```
    dx = layer1.backward(dp)

    layer1.update()

    if i % 100 == 0:
        print('epoch', i, 'loss', loss, 'y', y)

print(layer1.w)
print(layer1.b)
```

出力結果

```
epoch 0 loss 0.11059875819525353 y [[0.30140249 0.16058501 0.36526844 0.20329552]]
epoch 100 loss 0.07264510880272602 y [[0.22688731 0.27025334 0.39495912 0.45168125]]
epoch 200 loss 0.0527648595295762 y [[0.14751532 0.29044564 0.34971152 0.55988491]]
epoch 300 loss 0.0417154667463111 y [[0.09939268 0.28169867 0.30972685 0.61456522]]
epoch 400 loss 0.03449939597385088 y [[0.07045038 0.26694585 0.28070435 0.65218267]]
epoch 500 loss 0.029323894473012743 y [[0.05202754 0.25143319 0.25849141 0.68086392]]
epoch 600 loss 0.025403024800220022 y [[0.03971709 0.23684354 0.24062763 0.70394218]]
epoch 700 loss 0.022326384745484495 y [[0.03115796 0.22363927 0.22575441 0.72312334]]
epoch 800 loss 0.019851247873327545 y [[0.02500759 0.21184582 0.21307449 0.73941371]]
epoch 900 loss 0.017821589787150982 y [[0.02046322 0.20134291 0.20208214 0.75346842]]
[[2.62166652 2.61870924]]
[[-4.05594338]]
```

■ **考察**　本課題では AND 回路の入力と出力を使ってニューラルネットワークを学習させた．そして実行したあとのレイヤ 1 のウェイトとバイアスとそれらを使った出力を表示させた．しかし，結果を見たところ学習しきっているようには見えなかった．ただ，損失関数の値の下がり具合を見ると，学習途中のように見えたのでエポックを増やせばまだ損失は下がりそうだと感じた．

3.2 Exercise 3.5

■ **課題内容**　Check the values of weights and biases after learning in example3_3.m and write down these values to one places of decimals. Then, calculate XOR output by your hand calculation with step function.

■ **解答**　課題内容は XOR の入力と出力を使ってニューラルネットを学習させ，各レイヤのウェイトとバイアスを表示させ手作業で計算もするということだった．ソースコードと出力結果，手作業で計算した結果を以下に示す．

ソースコード 6　Exercise 3.5

```
x = np.array([[0, 0, 1, 1],
              [0, 1, 0, 1]])
t = np.array([0, 1, 1, 0])

input_dim = 2
hidden_dim = 2
output_dim = 1
```

```
8
9  w = 2.0 * np.random.rand(hidden_dim, input_dim) - 1.0
10 b = 2.0 * np.random.rand(hidden_dim, 1) - 1.0
11 u = 2.0 * np.random.rand(output_dim, hidden_dim) -1.0
12 c = 2.0 * np.random.rand(output_dim, 1) - 1.0
13
14 layer1 = Affine(w, b)
15 layer2 = Sigmoid()
16 layer3 = Affine(u, c)
17 layer4 = Sigmoid()
18 layer5 = MSE()
19
20 epoch = 1000
21 for i in range(epoch):
22     p = layer1.forward(x)
23     y = layer2.forward(p)
24     q = layer3.forward(y)
25     z = layer4.forward(q)
26     loss = layer5.forward(z, t)
27
28     dz = layer5.backward()
29     dq = layer4.backward(dz)
30     dy = layer3.backward(dq)
31     dp = layer2.backward(dy)
32     dx = layer1.backward(dp)
33
34     layer1.update(1.0)
35     layer3.update(1.0)
36
37     if i % 100 == 0:
38         print('epoch', i, 'loss', loss, 'z', z)
```

出力結果

```
epoch 0 loss 0.14865152481975835 z [[0.30973394 0.29240501 0.27691086 0.26406156]]
epoch 100 loss 0.12493005821637666 z [[0.51293776 0.50855688 0.49162716 0.4861852 ]]
epoch 200 loss 0.12465843369022318 z [[0.51062719 0.51422939 0.48651222 0.48670789]]
epoch 300 loss 0.12365131549983245 z [[0.51059522 0.53086825 0.47145759 0.47860356]]
epoch 400 loss 0.11606619291149047 z [[0.49631644 0.6006541  0.42374649 0.43663975]]
epoch 500 loss 0.0929762025847744 z [[0.45748439 0.74956163 0.39729043 0.32945326]]
epoch 600 loss 0.04646288191252964 z [[0.35850376 0.79248915 0.60704202 0.21377882]]
epoch 700 loss 0.009311833893073414 z [[0.14334216 0.86844748 0.8509094  0.12005676]]
epoch 800 loss 0.004461025559716306 z [[0.09645524 0.90192461 0.90219031 0.08484735]]
epoch 900 loss 0.0028436916336632635 z [[0.07621343 0.91943909 0.92374011 0.06808389]]
[[ 4.22786955 -4.5941637 ]
 [ 5.50647153 -5.78792623]]
[[-2.25110444]
 [ 3.17054042]]
[[ 7.18034835 -6.66447265]]
[[3.03835718]]
```

1000 エポック分学習させた状態で出力を手計算でステップ関数を適用させると，

$$\begin{bmatrix}0.1 & 0.9 & 0.9 & 0.1\end{bmatrix} = \begin{bmatrix}0 & 1 & 1 & 0\end{bmatrix} \qquad (\text{ステップ関数適用})$$

■ **考察**　本課題で XOR の入力と出力を学習させて，その出力を手計算でステップ関数を適用させた．まず，前提として資料に載っていた学習設定では学習しきっていなさそうだったため，学習率を 0.1 から 1.0 まで引き上げた．そうしたら 500 エポックを超え始めたあたりから急激に損失関数の出力が下がりはじめた様に見えた．また，実際に手作業でステップ関数も適用させた結果から正しく学習できているということが分かった．