# -\*- coding: utf-8 -\*-

2

3

# ----------------------------

4

# 必要なライブラリをインポート

5

# ----------------------------

6

import torch

7

import torch.nn as nn

8

9

10

# ----------------------------

11

# 2-1 準備

12

# ----------------------------

13

print("=======2-1 準備=======")

14

15

class SimpleMlp(nn.Module):

16

def \_\_init\_\_(self, vec\_length:int=16, hidden\_unit\_1:int=8, hidden\_unit\_2:int=2):

17

"""

18

引数:

19

vec\_length: 入力ベクトルの長さ

20

hidden\_unit\_1: 1つ目の線形層のニューロン数

21

hidden\_unit\_2: 2つ目の線形層のニューロン数

22

"""

23

# 継承しているnn.Moduleの\_\_init\_\_()メソッドの呼び出し

24

super(SimpleMlp, self).\_\_init\_\_()

25

# 1つ目の線形層

26

self.layer1 = nn.Linear(vec\_length, hidden\_unit\_1)

27

# 活性化関数のReLU

28

self.relu = nn.ReLU()

29

# 2つ目の線形層

30

self.layer2 = nn.Linear(hidden\_unit\_1, hidden\_unit\_2)

31

32

33

def forward(self, x: torch.Tensor) -> torch.Tensor:

34

"""順伝搬は、線形層→ReLU→線形層の順番

35

引数:

36

x: 入力。(B, D\_in)

37

B: バッチサイズ、 D\_in: ベクトルの長さ

38

返り値:

39

out: 出力。(B, D\_out)

40

B: バッチサイズ、 D\_out: ベクトルの長さ

41

"""

42

# 1つ目の線形層

43

out = self.layer1(x)

44

# ReLU

45

out = self.relu(out)

46

# 2つ目の線形層

47

out = self.layer2(out)

48

return out

49

50

vec\_length = 16 # 入力ベクトルの長さ

51

hidden\_unit\_1 = 8 # 1つ目の線形層のニューロン数

52

hidden\_unit\_2 = 2 # 2つ目の線形層のニューロン数

53

54

batch\_size = 4 # バッチサイズ。入力ベクトルの数

55

56

# 入力ベクトル。xの形状: (4, 16)

57

x = torch.randn(batch\_size, vec\_length)

58

# MLPを定義

59

net = SimpleMlp(vec\_length, hidden\_unit\_1, hidden\_unit\_2)

60

# MLPで順伝搬

61

out = net(x)

62

# MLPの出力outの形状が(4, 2)であることを確認

63

print(out.shape)