深度學習

HW1

學號 : B103012002

姓名 : 林凡皓

I. Creating and Accessing tensor :

● Creating_sample_tensor :

1. 解題思路 ： 這題我的想法是先創建一個 shape (2, 3)且元素皆為 0 的矩陣，再分別將位置(0, 1)和(1, 0)的值設定為 10 和 100。

● Mutate_tensor :

1. 解題思路 :透過 enumerate 取得 indices 中每一個座標（tuple）以及該座標在 indices 中的 index，再透過 for loop 將 values 中 index 位置上的值指派給 x 矩陣中對應座標上的值。
2. 額外討論 :本題題目有要求 mutate_tensor( )的參數 values 為一個存放浮點數的 list，但是在做作業時發現到如果 x 為一個元素皆為整數的矩陣，那 values 中含有浮點數的部分，經過 mutate_tensor( )後的新矩陣會自動將其變成整數，如圖(1)。因此在函數中，需要先將 x 中的所有元素都變成浮點數後再做運算才可以。

```
In [14]: x = torch.tensor([[1, 2, 3],
                           [4, 5, 6]])
         mutate_tensor(x, [(0, 0), (1, 2)], [100, 0.5])  #將x矩陣位置(0, 0)的值設定為100
                                                         #將x矩陣位置(1, 2)的值設定為0.5

Out[14]: tensor([[100,   2,   3],
                 [  4,   5,   0]])
```

圖(1)

● Count_tensor_elements :

1. 解題思路 :利用 x.shape 取的 x 的形狀，再透過 for loop 走訪 x.shape 中每個元素並相乘。
2. 額外討論 :本題需要先將 count_tensor_elements 的值設定成 1，否則執行相乘時會有問題。

● 執行結果 ：

```
Here is the sample tensor:
tensor([[  0.,  10.],
        [100.,   0.],
        [  0.,   0.]])

After mutating:
tensor([[ 4., 10.],
        [ 5.,  6.],
        [ 0.,  0.]])

Correct shape:  True
x[0, 0] correct:  True
x[1, 0] correct:  True
x[1, 1] correct:  True

Number of elements in x:  6
Correctly counted:  True
```

II. Tensor consturctors :

- ### Create_tensor_of_pi :

1. 解題思路 :本題可以先產生出形狀為(M, N)且元素皆為 1 的矩陣，再將矩陣中每個元素乘上 3.14 即可。

- ### 執行結果 :

```
x is a tensor: True
x has correct shape:  True
x is filled with pi:  True
```

III. Datatypes :

- ### Multiples_of_ten :

1. 解題思路 : 使用 for loop 將 start 和 stop 之間的數字走訪一遍並判斷此數除以 10 的餘數是否為 0。將以 10 的餘數為 0 的數字儲存在 list 中，利用此 list 產生 tensor，並且透過 dtype 將 tensor 中的元素設定成 torch.float64。
2. 額外討論 :本題有規定要取 start 和 stop 之間可被 10 整除的數字(包含 start 和 stop)，因此 range 的範圍要寫 stop + 1。

- ### 執行結果 :

```
Correct dtype:  True
Correct shape:  True
Correct values:  True

Correct dtype:  True
Correct shape:  True
```

IV. Slicing indexing :

- ### Slice_indexing_practice :

1. 解題思路 :Slice indexing 的規則為，x[range1, range2]，range1 為針對 row 的 slicing 範圍，range2 為針對 column 的 slicing 範圍 (range1、range2 範圍包含下限但不包含上限)。以下針對本題每一小題做詳細解釋。
   (1) last_row : 透過 x[-1]可完成 last_row。
   (2) third_row : 利用 slicing 範圍包含下限但不包含上限的特

性，可以透過 x[:, 2:3]取得 third_row

(3) first_two_rows_three_cols : first two rows 的範圍為 0:2, first three columns 的範圍為 0:3。

(4) even_rows_odd_cols : 利用範圍的第三個參數 (x:y:z 的 z) 來完成，此參數代表間隔大小。Even rows 可透過 0::2 (從 0 開始，每 2 格取一個數字，直到最後) 取得，odd columns 可透過 1::2 (從 1 開始，每 2 格取一個數字，直到最後) 取得

- **Slice_assignment_practice :**

1. 解題思路 :產生一個矩陣為([[0, 1, 2, 2, 2, 2],
   [0, 1, 2, 2, 2, 2],
   [3, 4, 3, 4, 5, 5],
   [3, 4, 3, 4, 5, 5]])

接著將此矩陣 assign 給 x[0:4, 0:6]。

- **執行結果 :**

1. Slice_indexing_practice :

```
last_row:
tensor([11, 12, 13, 14, 15])
Correct: True

third_col:
tensor([[ 3],
        [ 8],
        [13]])
Correct: True

first_two_rows_three_cols:
tensor([[1, 2, 3],
        [6, 7, 8]])
Correct: True

even_rows_odd_cols:
tensor([[ 2,  4],
        [12, 14]])
Correct: True
```

2. Slice_assignment_practice :

```
Here is x before calling slice_assignment_practice:
tensor([[0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0]])
Here is x after calling slice assignment practice:
tensor([[0, 1, 2, 2, 2, 2, 0],
        [0, 1, 2, 2, 2, 2, 0],
        [3, 4, 3, 4, 5, 5, 0],
        [3, 4, 3, 4, 5, 5, 0],
        [0, 0, 0, 0, 0, 0, 0]])
Correct:  True
```

V. Integer tensor indexing :

- **Shuffle_cols :**

  1. 解題思路 ;透過 slicing 的方式完成。針對 column 的 index，放入(0, 0, 2, 1)，代表說 y 的第一個 column 為 x 的第一個 column，y 的第二個 column 為 x 的第一個 column，y 的第三個 column 為 x 的第三個 column，y 的第四個 column 為 x 的第二個 column。

- **Reverse_rows :**

  1. 解題思路 :可透過內建函數 torch.flip(x, [0])來完成。torch.flip( )的第一個參數代表要做反轉的矩陣，第二個參數為要翻轉的維度，[0]為對 row 做翻轉，[1]為對 column 做翻轉。

- **Take_one_elem_per_col :**

  1. 解題思路 :要取得 x 矩陣中特定元素可以用 x[row_index, col_index]的語法取得，row_index 和 col_index 皆可放入 tuple 以便一次取得多個 x 矩陣中的元素。

- **Make_one_hot :**

  1. 解題思路 :先創建一個 shape(N, C)元素皆為 0 的矩陣，其中 N 為輸入矩陣 x 的長度，C 為 x 中元素最大值+1，要+1 的原因在於要考慮到 x 矩陣會有 0 的存在，而 0 會被放到輸出矩陣 y 的第一個 column(index = 0) 。接著可以透過 enumerate(x)取得 x 中每一個元素的 index 和 value，利用 for loop 將 y[index, value]設定成 1 即可。

- **執行結果 :**

  1. Shuffle_cols, reverse_rows, take_one_element_per_col :

```
Here is x:
tensor([[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9],
        [10, 11, 12]])

Here is shuffle_cols(x):
tensor([[ 1,  1,  3,  2],
        [ 4,  4,  6,  5],
        [ 7,  7,  9,  8],
        [10, 10, 12, 11]])
Correct: True

Here is reverse_rows(x):
tensor([[10, 11, 12],
        [ 7,  8,  9],
        [ 4,  5,  6],
        [ 1,  2,  3]])
Correct: True

Here is take_one_elem_per_col(x):
tensor([ 4,  2, 12])
Correct: True
```

2. Make_one_hot :

```
Here is y0:
tensor([[0., 1., 0., 0., 0.],
        [0., 0., 0., 0., 1.],
        [0., 0., 0., 1., 0.],
        [0., 0., 1., 0., 0.]])
y0 correct:  True

Here is y1:
tensor([[0., 1., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 1.],
        [0., 0., 0., 0., 0., 0., 1., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0.]])
y1 correct:  True
```

VI. Boolean tensor indexing :

- ## Sum_positive_entries :

1. 解題思路 :創建一個新 tensor 為 x[mask]，來取得 x 中所有大於 0 的元素。接著透過 torch.sum( )將 x 中所有大於 0 的元素相加。

- ## 執行結果 :

```
Correct for x0:  tensor(True)
Correct for x1:  tensor(True)
Correct for x2:  tensor(True)
```

VII. Reshaping operations :

- ## Reshape_practice :

1. 解題思路 :先利用 view( )將 x 變成 shape (2, 3, 4)的矩陣，接著過 torch.cat( )將新的 x 矩陣的兩個小矩陣，往 column 的方向合併。
2. 額外討論 :torch.cat( )可以傳入許多參數，這題只用到兩個。第一個為輸入一個 tuple，其中包含要合併的 tensor。第二個為要合併的 dimension，這題 dim = 1 因為要對 column 做擴張。

- ## 執行結果 :

```
Here is x:
tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23])
Here is y:
tensor([[ 0,  1,  2,  3, 12, 13, 14, 15],
        [ 4,  5,  6,  7, 16, 17, 18, 19],
        [ 8,  9, 10, 11, 20, 21, 22, 23]])
Correct: True
```

VIII. Tensor operations :

- ## Zero_row_min :

1. 解題思路 :可利用 torch.argmin( )來取得 x 矩陣中每一個 row 的最小值的 index。接著再透過 x.shape[0]取得 x 的 row 總數，並利用 torch.arange( )產生 y 矩陣的 row index。最後將最小值的元素設定成 0 即可。
2. 額外討論 :torch.argmin( )需要傳入兩個參數，第一個為 tensor，第二個為 dim，本題 dim = 1 因為要取得的是 column 的 index。torch.argmin( )會回傳一個 list。

- ## 執行結果 :

```
Here is x0:
tensor([[10, 20, 30],
        [ 2,  5,  1]])
Here is y0:
tensor([[ 0, 20, 30],
        [ 2,  5,  0]])
y0 correct:  True

Here is x1:
tensor([[ 2,  5, 10, -1],
        [ 1,  3,  2,  4],
        [ 5,  6,  2, 10]])
Here is y1:
tensor([[ 2,  5, 10,  0],
        [ 0,  3,  2,  4],
        [ 5,  6,  0, 10]])
y1 correct:  True
```

IX. Matrix operation :

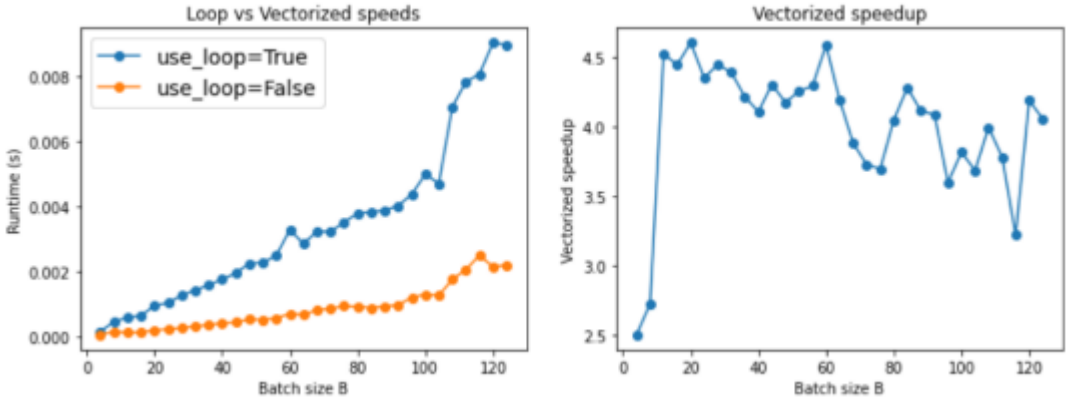
- ## Batched_matrix_multiply :

1. 解題思路 :
   (1) batched_matrix_multiply_loop :利用 x.shape[0]取得 x 矩陣第一維度的大小，透過 for loop 迭代，將 z[i]設定為 x[i]乘上 y[i]。矩陣乘法可以透過 torch.matmul( )完成。
   (2) batched_matrix_multiply_noloop :直接利用 torch.matmul( )將 x 矩陣與 y 矩陣相乘即可。

- ## 執行結果 :

```
z1 difference:  0.0
z1 difference within tolerance:  True

z2 difference:  4.76837158203125e-07
z2 difference within tolerance:  True
```



不使用迴圈會比使用迴圈快上許多。

X. Broadcasting :

- ## Normalize_columns :

1. 解題思路 :
   (1) 計算 mean : 透過 x.sum( )可以計算 x 中所有元素的和，如果傳入參數 dim = 0，即可求得每個 column 的元素和。再將求得的和除以 columnm 元素個數即可。
   (2) 計算 std : 計算(x – mean)^2，並透過 x.sum( )計算總合(dim = 0)，再將結果除以 column 元素個數(可透過 x.shpae[0]取得)開根號即可。
   (3) 標準化 : 計算(x – mean) / (std-1)即可完成標準化。

- ## 執行結果 :

```
Here is x:
tensor([[  0.,  30., 600.],
        [  1.,  10., 200.],
        [ -1.,  20., 400.]])
Here is y:
tensor([[ 0.,  1.,  1.],
        [ 1., -1., -1.],
        [-1.,  0.,  0.]])
y correct:  True
x unchanged:  True
```

XI. Running on GPU :

- ## mm_on_cpu & mm_on_gpu :

1. 解題思路 :先將輸入的矩陣放到 GPU 上，再利用 torch.matmul( )將兩個矩陣相乘，最後再用 y.cpu( )將結果矩陣放回 CPU。

- ## 執行結果 :

```
y1 on CPU: True
Max difference between y0 and y1: 0.004638671875
Difference within tolerance: True
CPU time: 38.25 ms
GPU time: 1301.72 ms
GPU speedup: 0.03 x
```

XII. Challenge :

- ## Challenge_mean_tensor :

1. 解題思路 :利用 torch.sum( )和 torch.stack( )將輸出矩陣 y 變成存放 x 中矩陣元素之和的 tensor。接著將 y 除以 ls 即可求得平均。在做除法前要先將 ls 變成 torch.tensor，因為 y 為 tensor，tensor 不能跟 list 做除法。
2. 執行結果 :

```
the first tensor in xs is tensor([1, 2, 3]), with mean = 2
the first tensor in xs is tensor([1, 1, 1, 2, 2, 2]), with mean = 1.5
the first tensor in xs is tensor([1, 1, 1, 2, 2, 2]), with mean = 2.5

result of challenge_mean_tensors(xs, ls) is tensor([2.0000, 1.5000, 2.5000])

the mean of the first tensor in xs is correct: True
the mean of the second tensor in xs is correct: True
the mean of the third tensor in xs is correct: True
```

3. 額外說明 :由於這題是加分題，因此測試的程式碼須自己寫，以下附上我自己寫的測試 code，以便確認這題的正確與否。

```
x1 = torch.tensor([1, 2, 3])              #x1's mean = 2
x2 = torch.tensor([1, 1, 1, 2, 2, 2])     #x2's mean = 1.5
x3 = torch.tensor([1.5, 2, 2.5, 3, 3.5])  #x3's mean = 2.5
xs = [x1, x2, x3]
ls = [3, 6, 5]

print(f"the first tensor in xs is {xs[0]}, with mean = 2")
print(f"the first tensor in xs is {xs[1]}, with mean = 1.5")
print(f"the first tensor in xs is {xs[1]}, with mean = 2.5")
print("")
print(f"result of challenge_mean_tensors(xs, ls) is {challenge_mean_tensors(xs, ls)}")
print("")
print(f"the mean of the first tensor in xs is correct: {challenge_mean_tensors(xs, ls)[0] == 2}")
print(f"the mean of the second tensor in xs is correct: {challenge_mean_tensors(xs, ls)[1] == 1.5}")
print(f"the mean of the third tensor in xs is correct: {challenge_mean_tensors(xs, ls)[2] == 2.5}")
```

- **Challenge_get_unique :**

1. 解題思路 :可透過 torch.argsort( )取得 x 的排序 index，並利用此 index 將 x 做排序。接著產生一個 mask，裡面存放 True 或是 False，代表說該 index 的數字是否唯一，並利用此 mask 取得唯一數字的 index。最後利用唯一數字的 index 取得唯一數字的值。
2. 執行結果 :

```
x = tensor([1, 2, 1, 2, 3, 2])
unique numbers are 1, 2, 3
the indices of the unique numbers are 0, 1, 4

Result of challenge_get_uniques:
unique numbers = tensor([1, 2, 3])
indices of the numbers are tensor([0, 1, 4])
```

3. 額外說明 :
   (1) mask 的建立 :比較相鄰的兩個數字，如果兩個數字相同，該 index 的值為 False，如果兩個值不相同，該 index 的值為 True。
   (2) 測試 code :

```
x = torch.tensor([1, 2, 1, 2, 3, 2])
print(f"x = {x}")
print(f"unique numbers are 1, 2, 3")
print(f"the indices of the unique numbers are 0, 1, 4\n")
print(f"Result of challenge_get_uniques: ")
print(f"unique numbers = {challenge_get_uniques(x)[0]}")
print(f"indices of the numbers are {challenge_get_uniques(x)[1]}")
```