

# 機器學習 HW1

資工碩一 M113040105 劉東霖

## 一. create\_sample\_tensor

### 1. 動機:

創立一個 tensor 並 assign tensor 內容某個值。

### 2. 過程:

(1). 首先，我會利用 `torch.zeros((3, 2))` 創立一個 3\*2 的 tensor **x**，此 tensor 的 element 全是 0。

(2). 指定 `x[0, 1]` 和 `x[1, 0]` 為 10 和 100。

### 3. 結果:

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

### 4. 程式碼:

```
x = torch.zeros((3, 2))  
x[0, 1], x[1, 0] = 10, 100
```

## 二. mutate\_tensor

### 1. 動機:

Assign indices 裡面的座標位置為 values 裡面的值。

### 2. 過程:

(1). 設定一個 count, 初始值為 0，為了要提取 values 裡面的數值。

(2). 用一個 for 迴圈: `for i0,i1 in indices:` 來提取每一個 tuples 的值。

(3). 利用剛剛的 `i0` 和 `i1` 將指定的 index 換上指定的 values，如下圖：

```
count=0
for i0,i1 in indices:
    x[i0,i1]=values[count]
```

(4). `count+=1` 讀下一個 values。

3. 執行結果：

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

4. 程式碼：

```
count=0
for i0,i1 in indices:
    x[i0,i1]=values[count]
    count=count+1
```

### 三. `count_tensor_element`

1. 動機：

計算 tensor 的總 element 數。

2. 過程：

(1). 因為題目的 tensor 的 shape 是二維的，所以

`total_element` 我直接用 `num_element=x.shape[0]*x.shape[1]`，然後回傳。

3. 執行結果：

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

#### 4. 程式碼:

```
num_elements=x.shape[0]*x.shape[1]
```

### 四.create\_tensor\_of\_pi

#### 1. 動機:

把 tensor 的值改成 pi。

#### 2. 過程:

(1). 利用 `x=torch.zeros(M, N)` 創立一個 shape 為 (M, N) 的 tensor, element 全是 0。

(2). 利用 `x[:, :]=3.14` 把所有 element 都變成 3.14, 並回傳 x。

#### 3. 執行結果:

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

#### 4. 程式碼:

```
x = torch.zeros(M, N)
x[:, :]=3.14
```

### 五.multiples\_of\_ten

#### 1. 動機:

把 tensor 裡 10 的餘數的 element 放出來。

#### 2. 過程:

(1). 利用一個 for loop 把 10 的餘數的內容放到 x 陣列裡

面，程式碼如下圖：

```
x=[i for i in range(start,stop,1) if i%10==0]
```

(2). 把 x 轉成 tensor，data\_type 轉成 float64 並回傳。

3. 執行結果：

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

4. 程式碼：

```
x=[i for i in range(start,stop,1) if i%10==0]  
x=torch.tensor(x, dtype=torch.float64)
```

## 六. slice\_indexing\_practice

1. 動機：把 tensor 某幾個欄位抓出來。

2. 過程：

原 tensor：

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

(1). 利用 `x[-1,:]` 把最後一個 row 的 element 存起來，如下

圖：

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

(2). 利用 `x[:, 2:3]` 把中間 column 的 element 存起來，如

下圖：

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

(3). 利用 `x[:2, :3]` 把前兩個 row 裡的前 3 個 column 的

element 存起來，如下圖：

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

(4). 利用 `x[0::2, 1::2]` 把奇數 row 裡面的偶數 column 裡

的 element 存起來，如下圖：

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

3. 執行結果：

```
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
```

#### 4. 程式碼:

```
last_row = x[-1,:]  
third_col = x[:,2:3]  
first_two_rows_three_cols = x[:2,:3]  
even_rows_odd_cols = x[0::2,1::2]
```

### 七. slice\_assignment\_practice

#### 1. 動機:

Assign tensor 的某些欄位某個值。

#### 2. 過程:

原 tensor:

```
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]])
```

(1). 利用 `x[0:2, 1:2]=1` 把第 0, 1 個 row 裡面的第一個 column 的 element 改成 1，修改的地方如下圖：

```
tensor([[0, 1, 0, 0, 0, 0, 0],  
        [0, 1, 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]])
```

(2). 利用 `x[0:2, 2:6]=2` 把第 0, 1 個 row 裡面的第 2~5 個 column 的 element 改成 2，修改的地方如下圖：

```
tensor([[0, 0, 2, 2, 2, 2, 0],  
        [0, 0, 2, 2, 2, 2, 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]])
```

(3). 利用 `x[2:4, 0:4:2]=3` 把第 2, 3 個 row 裡面的第 0, 2 個 column 的 element 改成 3，修改的地方如下圖：

(4). 利用 `x[2:4, 1:4:2]=4` 把第 2, 3 個 row 裡面的第 1, 3 個

```
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]])
```

column 的 element 改成 4，修改的地方如下圖：

```
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]])
```

(5). 利用 `x[2:4, 4:6]=5` 把第 2, 3 個 row 裡面的第 4, 5 個

column 的 element 改成 5，修改的地方如下圖：

```
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]])
```

### 3. 執行結果：

```
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
```

### 4. 程式碼：

```
x[0:2, 1:2]=1
x[0:2, 2:6]=2
x[2:4, 0:4:2]=3
x[2:4, 1:4:2]=4
x[2:4, 4:6]=5
```

## 八. shuffle\_cols

### 1. 動機:

新增維度且 assign tensor 的某些欄位某個值。

### 2. 過程:

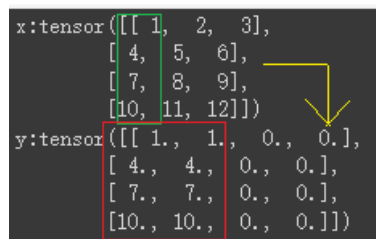
原 tensor:

```
[[ 1,  2,  3],  
 [ 4,  5,  6],  
 [ 7,  8,  9],  
 [10, 11, 12]]
```

(1). 利用 `y=torch.zeros(x.shape[0], x.shape[1]+1)`，創建一個 row 為 `x.shape[0]`，column 為 `x.shape[1]+1` 的 tensor，element 全是 0。

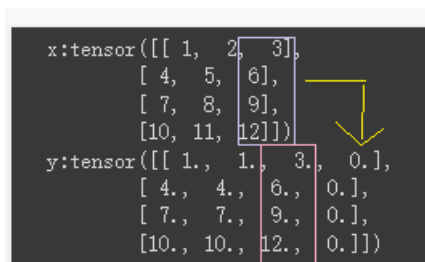
(2). 利用 `y[:, 0:2]=x[:, 0:1]` 把 y 的第 0, 1 個 column 的 element 複製 x 的第 0 個 column，如圖下所示:

```
x:tensor([[ 1,  2,  3],  
          [ 4,  5,  6],  
          [ 7,  8,  9],  
          [10, 11, 12]])  
y:tensor([[ 1.,  1.,  0.,  0.],  
          [ 4.,  4.,  0.,  0.],  
          [ 7.,  7.,  0.,  0.],  
          [10., 10.,  0.,  0.]])
```



(3). 利用 `y[:, 2]=x[:, 2]` 把 y 的第 2 個 column 複製 x 的第 2 個 column，如圖下所示:

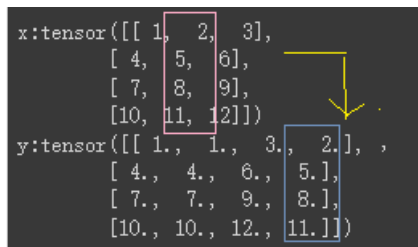
```
x:tensor([[ 1,  2,  3],  
          [ 4,  5,  6],  
          [ 7,  8,  9],  
          [10, 11, 12]])  
y:tensor([[ 1.,  1.,  3.,  0.],  
          [ 4.,  4.,  6.,  0.],  
          [ 7.,  7.,  9.,  0.],  
          [10., 10., 12.,  0.]])
```





(4). 利用  $y[:, 3]=x[:, 1]$  把 y 的第 3 個 column 複製 x 的第

1 個 column，如圖下所示：



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

3. 執行結果：

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

4. 程式碼：

```
y=torch.zeros(x.shape[0], x.shape[1]+1)
y[:, 0:2]=x[:, 0:1]
y[:, 2]=x[:, 2]
y[:, 3]=x[:, 1]
```

## 九. Reverse\_rows

1. 動機：

把 tensor 原本 row 的排列倒轉。

2. 過程：

原 tensor

```
[[ 1,  2,  3],
 [ 4,  5,  6],
 [ 7,  8,  9],
 [10, 11, 12]]
```

(1). 利用 `indices= torch.arange(x.shape[0]-1, -1, -1)` 創建一個 list，倒轉時可以用。假設 `x.shape[0]=4`, 此 list 的內容為 `[3, 2, 1, 0]`。

(2). 利用 `y=x.clone()` 讓 `y=x`。

(3). 利用 `y[:, :]=x[indices, :]` 來達成我倒轉 row 的目的。

3. 執行結果:

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

4. 程式碼:

```
indices=torch.arange(x.shape[0]-1,-1,-1)
y = x.clone()
y[:, :]=x[indices, :]
```

## 十. take\_one\_elem\_per\_col

1. 動機:

新增 tensor 且內容為原本 tensor 的某些內容。

2. 過程:

原 tensor

```
[[ 1,  2,  3],
 [ 4,  5,  6],
 [ 7,  8,  9],
 [10, 11, 12]]
```

(1). 利用 `y=torch.tensor([x[1, 0], x[0, 1], x[3, 2]])` 把

`x[1, 0], x[0, 1], x[3, 2]` 變成一個 tensor, 並回傳。

### 3. 執行結果：

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

### 4. 程式碼：

```
y=torch.tensor([x[1,0],x[0,1],x[3,2]])
```

## 十一. Make\_one\_hot

### 1. 動機：

新增 tensor 並創造特殊的 one hot tensor。

### 2. 過程：

(1). 利用  $N=\text{len}(x)$  算出 x list 的長度。

(2). 利用  $\text{find\_max}=\text{torch.tensor}(x)$  把 x 轉成 tensor。

(3). 利用  $C=\text{torch.max}(\text{find\_max})+1$ ，來找出 x 裡最大值，

此 c 為輸出 tensor 的 column 數。

(4). 利用  $y=\text{torch.zeros}(N,C)$  創建一個  $N \times C$  的 tensor，  
element 全為 0。

(5). 利用  $\text{index}=\text{range}(y.\text{shape}[0])$  來創建一個 list，內容  
視  $y.\text{shape}[0]$  (也等於 N) 決定，

如果  $y.\text{shape}[0]$  為 4,  $\text{index}=[0, 1, 2, 3]$ 。

(6). 利用 `y[index, x]=1` 把 `index` 對應到 `x` 的內容為 1。假設 `index=[0, 1, 2, 3]`，`x=[1, 4, 3, 2]`，`y[0, 1]`和 `y[1, 4]`和 `y[2, 3]`和 `y[3, 2]=1`。

3. 執行結果：

```
x0 = [1, 4, 3, 2]
y0 = make_one_hot(x0)
print('Here is y0:')
print(y0)
print('y0 correct: ', check_one_hot(x0, y0))

x1 = [1, 3, 5, 7, 6, 2]
y1 = make_one_hot(x1)
print('\nHere is y1:')
print(y1)
print('y1 correct: ', check_one_hot(x1, y1))
```

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

4. 程式碼：

```
N=len(x)
find_max=torch.tensor(x)
C=torch.max(find_max)+1
y = torch.zeros(N,C)
index=range(y.shape[0])
y[index, x]=1
```

## 十二. `sum_positive_entries`

1. 動機：

計算 tensor 裡  $>0$  的 element 有幾個。

## 2. 過程:

(1). 利用 `pos_value=x[x>0]` 創一個 tensor，假如 x tensor 裡面的內容  $>0$ ，`pos_value` 裡的內容為原本的值，否則為 0。

(2). 利用 `pos_sum=torch.sum(pos_value)` 相加 tensor 裡面  $>0$  的數，並回傳。

## 3. 執行結果

```
x0 = torch.tensor([[[-1, -1, 0], [0, 1, 2], [3, 4, 5]])
x1 = torch.tensor([-100, 0, 1, 2, 3])
x2 = torch.randn(100, 100).long()
print('Correct for x0: ', sum_positive_entries(x0) == 15)
print('Correct for x1: ', sum_positive_entries(x1) == 6)
print('Correct for x2: ', sum_positive_entries(x2) == 1871)

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

## 4. 程式碼:

```
pos_value=x[x>0]
pos_sum=torch.sum(pos_value)
```

# 十三. Reshape\_practice

## 1. 動機:

透過一連串的 `reshape`，`permute`，`view`，`transpose` 來改變矩陣。

## 2. 過程:

原本 tensor

```
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])
```

(1). 利用 `y=x.view(2,12)` 把 tensor shape 變(2,12)，如下

圖所示：

```
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]])
```

(2). 利用 `y=x.transpose(0,1)` 打亂記憶體空間，把 tensor shape 變(12,2)，如下圖所示：

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

(3). 利用 `y=x.reshape(3,4,2)` 把 tensor shape 變

(3,4,2)，讓題目答案所期望的形式快出來了。如下圖所示：

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

(4). 利用 `y=x.permute(0,2,1)` 把 tensor shape 變

(3,2,4)。如下圖所示：

```
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]]])
```

(5) . 利用 `y=x.reshape(3, 8)`把 tensor shape 變(3, 8) ,  
變成題目所要的格式。

### 3. 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
```

### 4. 程式碼:

```
# replace pass statement with your code
y=x.view(2,12)
y=y.transpose(0,1)
#[[0,12], [1,13], [2,14],....]
y=y.reshape(3,4,2)
#[[[0,12], [1,13], [2,14], [3,15]], [[4,16], [5,17], [6,18], [7,19]]....]
y=y.permute(0,2,1)
y=y.reshape(3,8)
```

## 十四. Zero\_min\_rows

### 1. 動機:

把每一個 row 裡面最小的 element 拿出來。

### 2. 過程:

#### (1). 利用

`min_vals,min_idx=torch.min(x,dim=1,keepdim=True)`找到每一個  
row 裡面的最小值和位置，並用 `keepdim=True` 讓他的 shape 和 x 一  
樣。

#### (2). 利用 `y=x.clone()`讓 `y=x`。

(3). 利用 `y[y==min_val]=0` 讓每一個 row 的最小值為 0。

3. 執行結果：

```
Here is x0:
tensor([[10, 20, 30],
        [ 2,  5, 11]])
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
```

4. 程式碼：

```
min_vals, min_idxs=torch.min(x, dim=1, keepdim=True)
y=x.clone()
y[y==min_vals]=0
```

## 十五. batch\_matrix\_multiply

1. 動機：

比較兩個不同 batch matrix multiply 的差異。

2. 過程：

1. use loop:

(1). 建立輸出 tensor 的格式。大小為

`x.shape[0], x.shape[1], y.shape[2]`



(2). 用一個 for 來解決 batch 的矩陣相乘，把 `x.shape[0]` 當成 batch 數，如下圖所示：

```
z=torch.zeros(x.shape[0],x.shape[1],y.shape[2])
for i in range(x.shape[0]):
    z[i]=torch.mm(x[i],y[i])
```

2. use `torch.bmm(x,y)`: 解決 `batch_matrix_multiply`。

3. 執行結果：

```
z1 = batched_matrix_multiply(x, y, use_loop=True)
z1_diff = (z1 - z_expected).abs().max().item()
print('z1 difference: ', z1_diff)
print('z1 difference within tolerance: ', z1_diff < 1e-6)

z2 = batched_matrix_multiply(x, y, use_loop=False)
z2_diff = (z2 - z_expected).abs().max().item()
print('\nz2 difference: ', z2_diff)
print('z2 difference within tolerance: ', z2_diff < 1e-6)

z1 difference: 0.0
z1 difference within tolerance: True

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

## 十六. `normalize_columns`

1. 動機：

把 tensor 裡的每一個 column 做 standard normalization。

2. 過程：

(1). 使用公式：

$$\mu = \frac{1}{M} \sum_{i=1}^M x_i \quad \sigma = \sqrt{\frac{1}{M-1} \sum_{i=1}^M (x_i - \mu)^2}$$

(2). 利用 `mean=torch.sum(x, dim=0)/x.shape[0]` 算出 column 的平均值。

(3). 利用 `sigma=torch.sqrt(torch.sum((x-mean)**2, dim=0)/(x.shape[0]-1))` 算出 column 的標準差。

(4). 利用 `y=(x-mean)/sigma` 做 normalization。

### 3. 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
```

### 4. 程式碼:

```
mean=torch.sum(x, dim=0)/x.shape[0]
sigma=torch.sqrt(torch.sum((x-mean)**2, dim=0)/(x.shape[0]-1))
y=(x-mean)/sigma
```

## 十七. mm\_on\_cpu\_and\_gpu

### 1. 動機:

比較矩陣相乘在 cpu 和 gpu 上計算時間的差異。

### 2. 過程:

#### 1. on\_cpu

(1). 利用 `x=x.to('cpu')` 把 x 放在 cpu 執行。

- (2). 利用 `w=w.to('cpu')` 把 `w` 放在 `cpu` 執行。
- (3). 利用 `y=torch.matmul(x,w)` 把 `x` 和 `w` 做矩陣相乘。
- (4). 利用 `y=y.to('cpu')` 把 `y` 放回 `cpu`。

## 2. on\_gpu

- (1). 利用 `x=x.to('cuda')` 把 `x` 放在 `gpu` 執行。
- (2). 利用 `w=w.to('cuda')` 把 `w` 放在 `gpu` 執行。
- (3). 利用 `y=torch.matmul(x,w)` 把 `x` 和 `w` 做矩陣相乘。
- (4). 利用 `y=y.to('cpu')` 把 `y` 放回 `cpu`，這樣做的原

因是後面要比較兩個裝置上的 speedup，裝置要一樣，所以我把運算完的東西放到 `cpu`。

## 3. 執行結果：

```
y1 on CPU: True
Max difference between y0 and y1: 0.001220703125
Difference within tolerance: True
CPU time: 226.80 ms
GPU time: 28.05 ms
GPU speedup: 8.08 x
```

## 4. 程式碼：

```
x=x.to('cpu')
w=w.to('cpu')
y = x.mm(w)
y=y.to('cpu')
```

```
x=x.to('cuda')
w=w.to('cuda')
y = torch.matmul(x,w)
y=y.to('cpu')
```