AIMS 5702 Artificial Intelligence in Practice (Fall 2025-26)

AIMS 5702 Assignment 1

Please complete this colab and:

- Download it as **both** .ipynb file and .pdf file (click on file -> download -> download as .ipynb file)
- Submit **both** .pdf and .ipynb file to blackboard

No GPU is required. Just use CPU host time is enough.

Deadline of submission: 23:59, Sept. 14th (Sunday), 2025

Import

```
#@title Import

import torch
import math
import numpy as np
from typing import Optional
import scipy.interpolate
```

Utilities

```
#@title Utilities

def is_same_tensor(result: torch.Tensor,
                   ref: torch.Tensor,
                   tol: Optional[float]=None) -> bool:
    """
    Check if two tensors are the same.

    Args:
        result: Results by your code.
        ref: Ground truth result.

    Return:
        Whether result and ref are the same.
    """
    if (not isinstance(result, torch.Tensor) or
            not isinstance(ref, torch.Tensor)):
        return False
    if result.dtype != ref.dtype:
        result = result.to(ref.dtype)
    if tol is not None:
        return torch.allclose(result, ref, rtol=0, atol=tol)
    else:
        return torch.equal(result, ref)
```

Question 1. Matrix Multiple

Recall the in pytorch, matrix multiple is using torch.matmal. In this assignment, we will implement them without using calling torch.matmul (or use @), but implement them by yourselves.

We ask you to implement it in two ways.

First, implement it using two-nested forloop (exactly two nested forloop).

Hint: you need torch.sum.

Q1.1. matmul_forloop

```
#@title Q1.1. matmul_forloop

def matmul_forloop(
        x: torch.Tensor,
        y: torch.Tensor
) -> torch.tensor:
    """
    Using python forloop to implement torch.matmul.

    Args:
        x: First matrix.
        y: Second matrix.

    Returns:
        Result matrix. If two input do not match, return None.
    """
    # Check if inputs are 2D tensors
    if len(x.shape) != 2 or len(y.shape) != 2:
            return None

    #Get shape from Matrix
    m, n = x.shape
    p, q = y.shape

    # Check if shapes are compatible
    if n != p:
            return None

    # Initialize result tensor
    result = torch.zeros((m, q), dtype=x.dtype, device=x.device)

    # Implement matrix multiplication using exactly two nested for loops
    for i in range(m):
            for j in range(q):
                    # Use torch.sum to compute the dot product
                    result[i, j] = torch.sum(x[i, :] * y[:, j])

    return result
```

Second, implement it using vector/matrix operations. No forloop statement is allowed this time.

Hints: use torch.sum and broadcast.

Q1.2. matmul_nofor

```
#@title Q1.2. matmul_nofor

def matmul_nofor(
        x: torch.tensor,
        y: torch.tensor
) -> torch.tensor:
    """
    Using pytorch vector operations to implement torch.matmul. No forloop is
    allowed.

    Args:
        x: First matrix.
        y: Second matrix.

    Returns:
        Result matrix. If two input do not match, return None.
    """

    # Check if inputs are 2D tensors
    if len(x.shape) != 2 or len(y.shape) != 2:
            return None

    #Get shape from Matrix
    m, n = x.shape
    p, q = y.shape

    # Check if shapes are compatible
    if n != p:
            return None

    # Expand x to (m, n, 1), y to (1, n, q), multiply and sum over dim=1
    x_expanded = x.unsqueeze(2)   # shape: (m, n, 1)
    y_expanded = y.unsqueeze(0)   # shape: (1, n, q)
    product = x_expanded * y_expanded   # broadcast multiply: (m, n, q)
    result = torch.sum(product, dim=1)   # sum over the second dimension: (m, q)

    return result
```

Third, implement it using einsum.

Q1.3. matmul_einsum

```
#@title Q1.3. matmul_einsum

def matmul_einsum(
        x: torch.tensor,
        y: torch.tensor
) -> torch.tensor:
    """
    Using pytorch vector operations to implement torch.matmul. No forloop is
    allowed.

    Args:
        x: First matrix.
        y: Second matrix.

    Returns:
        Result matrix. If two input do not match, return None.
    """

    # Check if inputs are 2D tensors
    if len(x.shape) != 2 or len(y.shape) != 2:
        return None

    m, n = x.shape
    p, q = y.shape

    # Check if shapes are compatible
    if n != p:
        return None

    # Implement matrix multiplication using torch.einsum
    result = torch.einsum('ij,jk->ik', x, y)

    return result
```

Testing

Please do run this block before submission, but do not change it. If you change it, you will get no mark for this question.

Test 1

```
#@title Test 1

dim1list = [2, 3, 10, 30]
dim2list = [2, 1, 5, 100]
dim3list = [2, 2, 10, 100]

torch.manual_seed(1234)
for i, (dim1, dim2, dim3) in enumerate(zip(dim1list, dim2list, dim3list)):
    a = torch.randint(0, 100, size=(dim1, dim2))
    b = torch.randint(0, 100, size=(dim2, dim3))
    c_ref = a @ b
    c_forloop = matmul_forloop(a, b)
    c_nofor = matmul_nofor(a, b)
    c_einsum = matmul_einsum(a, b)
    assert is_same_tensor(c_ref, c_forloop)
    assert is_same_tensor(c_ref, c_nofor)
    assert is_same_tensor(c_ref, c_einsum)
    print(f'{i}-th test succeeds')
```

```
0-th test succeeds
1-th test succeeds
2-th test succeeds
3-th test succeeds
```

Test 2

```
#@title Test 2

dim1 = 500
dim2 = 1000
dim3 = 2000

torch.manual_seed(1234)
a = torch.rand(size=(dim1, dim2), dtype=torch.float32)
b = torch.rand(size=(dim2, dim3), dtype=torch.float32)
c_ref = a @ b
c_nofor = matmul_nofor(a, b)
c_einsum = matmul_einsum(a, b)
assert is_same_tensor(c_ref, c_nofor, 1e-3)
assert is_same_tensor(c_ref, c_einsum, 1e-3)
print('test succeeds')
```

```
test succeeds
```

Test 3

```
#@title Test 3

a = torch.rand(size=(3, 5))
b = torch.rand(size=(3, 5, 7))
assert matmul_forloop(a, b) == None
assert matmul_nofor(a, b) == None
assert matmul_einsum(a, b) == None

a = torch.rand(size=(3, 5))
b = torch.rand(size=(3, 5))
assert matmul_forloop(a, b) == None
assert matmul_nofor(a, b) == None
assert matmul_einsum(a, b) == None

a = torch.rand(size=(3, 5))
b = torch.rand(size=(7, 4))
assert matmul_forloop(a, b) == None
assert matmul_nofor(a, b) == None
assert matmul_einsum(a, b) == None

print('test succeeds')
```

```
test succeeds
```

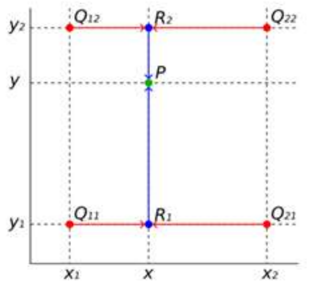
Question 2. Bilinear Interpolation

Next, let us try to implement 2D bilinear interpolation.

Details of 2D linear interpolation algorithm is described in:

https://en.wikipedia.org/wiki/Bilinear_interpolation

Basically, it defines a 2D function using grid data x and y, and its actual value on each grid points.



Then given a query x0, y0, the interpolation algorithm shall calculate the interped value.

$$f(x, y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}$$

For your information, here is a simple forloop implementation.

Reference implementation.

```
#@title Reference implementation.

def interp2d_forloop(
        v: torch.Tensor,
        x_ref_list: torch.Tensor,
        y_ref_list: torch.Tensor,
) -> torch.Tensor:
    # We need to do input check, but here I deliberately skip it.
    h, w = v.shape
    nlen = x_ref_list.shape[0]
    v_ref = torch.zeros(nlen, dtype=torch.float32)

    for i in range(nlen):
        x_ref = x_ref_list[i]
        y_ref = y_ref_list[i]
        x1 = int(max(math.floor(x_ref), 0))
        y1 = int(max(math.floor(y_ref), 0))
        x2 = int(min(x1 + 1, w - 1))
        y2 = int(min(y1 + 1, h - 1))
        dx2 = x2 - x_ref
        dx1 = 1 - dx2
        dy2 = y2 - y_ref
        dy1 = 1 - dy2
        f11 = v[y1, x1]
        f12 = v[y1, x2]
        f21 = v[y2, x1]
        f22 = v[y2, x2]
        v_ref[i] = (f11 * dy2 * dx2 + f12 * dy2 * dx1  +
                        f21 * dy1 * dx2 + f22 * dy1 * dx1)
    return v_ref
```

Now, it is your turn. Implement it using pytorch, but without using forloop.

Q2.1. interp2d_forloop

```
#@title Q2.1. interp2d_forloop

def interp2d_nofor(
        v: torch.Tensor,
        x_ref_list: torch.Tensor,
        y_ref_list: torch.Tensor,
) -> torch.Tensor:
    """
    Implement interp2d using python, without forloop and using 3rd-party library.

    Args:
        v: value matrix, which defines v(x, y). x and y are grid points. 2D tensor.
        x_ref: list of x where to grap interpolated value. 1D tensor.
        y_ref: list of y where to grap interpolated value. 1D tensor.

    Returns:
        Interpolation results. 1D tensor
    """
    # Input checks: Ensure v is 2D, x_ref_list and y_ref_list are 1D and have the same length
    if len(v.shape) != 2 or len(x_ref_list.shape) != 1 or len(y_ref_list.shape) != 1:
            return None
    if x_ref_list.shape[0] != y_ref_list.shape[0]:
            return None

    h, w = v.shape
    nlen = x_ref_list.shape[0]

    # Compute x1, y1, x2, y2
    x1 = torch.floor(x_ref_list).long().clamp(0, w - 1)
    y1 = torch.floor(y_ref_list).long().clamp(0, h - 1)
    x2 = (x1 + 1).clamp(0, w - 1)
    y2 = (y1 + 1).clamp(0, h - 1)

    # Compute weights
    dx2 = x2.float() - x_ref_list
    dx1 = 1 - dx2
    dy2 = y2.float() - y_ref_list
    dy1 = 1 - dy2

    # Extract values from the four corner points (vectorized indexing)
    f11 = v[y1, x1]
    f12 = v[y1, x2]
    f21 = v[y2, x1]
    f22 = v[y2, x2]

    # Compute bilinear interpolation result
    v_ref = (f11 * dy2 * dx2 + f12 * dy2 * dx1 +
                f21 * dy1 * dx2 + f22 * dy1 * dx1)

    return v_ref
```

Testing

Please do run this block before submission, but do not change it. If you change it, you will get no mark for this question.

Test 1

```
#@title Test 1

torch.manual_seed(1234)

h_test_list = []
h = 100
w = 50
vlen = 30
v = torch.rand(h, w).to(torch.float32)
x = torch.linspace(0, w-1, w)
y = torch.linspace(0, h-1, h)

interp = scipy.interpolate.RegularGridInterpolator(
        (y.numpy(), x.numpy()), v.numpy(), method='linear')
x_ref = torch.rand(vlen) * (w - 1)
y_ref = torch.rand(vlen) * (h - 1)
xy_ref = np.stack((y_ref.numpy(), x_ref.numpy()), axis=1)
v_ref = interp(xy_ref)

v_ref_forloop = interp2d_forloop(v, x_ref, y_ref)
v_ref_nofor = interp2d_nofor(v, x_ref, y_ref)

assert is_same_tensor(torch.tensor(v_ref), v_ref_forloop, tol=1e-2)
assert is_same_tensor(torch.tensor(v_ref), v_ref_nofor, tol=1e-2)
print('test succeeds')
```

```
test succeeds
```

Test 2

```
#@title Test 2

v = torch.rand(10, 4)
x_ref = torch.tensor([2])
y_ref = torch.tensor([2, 4])
assert interp2d_nofor(v, x_ref, y_ref) == None

v = torch.rand(10, 4)
x_ref = torch.tensor([[2, 3], [3, 4]])
x_ref = torch.tensor([[2, 3], [3, 4]])
assert interp2d_nofor(v, x_ref, y_ref) == None

print('test succeeds')
```

```
test succeeds
```