Task 1 - Broadcasted Distance Calculation

- Objective: Compute pairwise Euclidean distances between two sets of points **without loops**.

Here's the Euclidean distance formula in Markdown format:

Euclidean Distance Formula

The Euclidean distance between two points

$$p = (p_1, \ldots, p_n)$$

and

$$q = (q_1, \ldots, q_n)$$

in n-dimensional Euclidean space is defined as:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

For 2D Space (x-y plane)

In two-dimensional space, for points

$$p = (x_1, y_1)$$

and

$$q = (x_2, y_2)$$

, the formula simplifies to:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This formula calculates the straight-line distance between two points by taking the square root of the sum of squared differences of their coordinates.

Task Details:

Inputs:

- X: a 2D array of shape (N, D) for N points in D-dimensional space.
- Y: a 2D array of shape (M, D) for M points in D-dimensional space.

Output:

- dist: a 2D array of shape (N, M), where dist[i, j] is the Euclidean distance between X[i] and Y[j].

Hint: Use broadcasting

Example

```
import numpy as np

X = np.array([[0, 0],
              [1, 1],
              [2, 2]])  # Shape (3, 2)

Y = np.array([[0, 0],
              [0, 1],
              [1, 1],
              [2, 2]])  # Shape (4, 2)

# Expected dist array:
# Distances between:
# When X[0]=(0,0) and Y: (0,0) distance will be 0,  and when Y: (0,1) distance will be 1, and when Y: (1,1) distance will be
```

```
# X[1]=(1,1), Y: (0,0)=√2, (0,1)=1,      (1,1)=0,        (2,2)=√2≈1.4142
# X[2]=(2,2), Y: (0,0)=√8, (0,1)=√5≈2.2361, (1,1)=√2≈1.4142, (2,2)=0

# dist should be approximately:
# [[0.        , 1.        , 1.41421356, 2.82842712],
#  [1.41421356, 1.        , 0.        , 1.41421356],
#  [2.82842712, 2.23606798, 1.41421356, 0.        ]]
```

```
import numpy as np

def calculate_distances(X, Y):
    # Compute squared differences using broadcasting
    diff = X[:, np.newaxis, :] - Y[np.newaxis, :, :]

    # Square differences and sum along last axis
    squared_distances = np.sum(diff**2, axis=2)

    # Take square root to get Euclidean distances
    distances = np.sqrt(squared_distances)

    return distances

# Example inputs
X = np.array([[0, 0],
              [1, 1],
              [2, 2]])  # Shape (3, 2)

Y = np.array([[0, 0],
              [0, 1],
              [1, 1],
              [2, 2]])  # Shape (4, 2)

# Calculate distances
dist = calculate_distances(X, Y)

# Print result
print(dist)
```

```
[[0.         1.         1.41421356 2.82842712]
 [1.41421356 1.         0.         1.41421356]
 [2.82842712 2.23606798 1.41421356 0.        ]]
```

Task 2- Row-wise and Column-wise Normalization

- Objective: Normalize rows and columns of a matrix so that each row or column sums to 1.

Task Details:

Input:

- A: a 2D array (N, M).

Outputs:

- A_row_norm: same shape as A, each row sums to 1.
- A_col_norm: same shape as A, each column sums to 1. Hint:

Row normalization: A_row_norm = A / A.sum(**put your params here**) Column normalization: A_col_norm = A / A.sum(**put your params here**)

```
A = np.array([[1, 2],
              [3, 4]])
```

```
# Row sums: [3, 7]
# A_row_norm:
# [[1/3, 2/3],
#  [3/7, 4/7]]
# ≈ [[0.33333333, 0.66666667],
#    [0.42857143, 0.57142857]]

# Column sums: [4, 6]
# A_col_norm:
# [[1/4, 2/6],
#  [3/4, 4/6]]
# = [[0.25, 0.33333333],
#    [0.75, 0.66666667]]
```

```
import numpy as np

def normalize_matrix(A):
    # Row-wise normalization: Normalize each row so that sum of elements in each row is 1.
    A_row_norm = A / A.sum(axis=1, keepdims=True)

    # Column-wise normalization: Normalize each column so that sum of elements in each colu
    A_col_norm = A / A.sum(axis=0, keepdims=True)

    return A_row_norm, A_col_norm

# Example input
A = np.array([[1, 2],
              [3, 4]])

# Normalize
A_row_norm, A_col_norm = normalize_matrix(A)

# Print results
print("Row-wise Normalization:\n", A_row_norm)
print("\nColumn-wise Normalization:\n", A_col_norm)
```

```
Row-wise Normalization:
 [[0.33333333 0.66666667]
 [0.42857143 0.57142857]]

Column-wise Normalization:
 [[0.25       0.33333333]
 [0.75       0.66666667]]
```

Task 3- Elementwise Condition Replacement

- Objective: Replace elements based on conditions without loops.

Task Details:

Input:

- x: a 1D array.
- T: a threshold.

Steps:

- Replace all negative values in x with 0.
- Replace all values greater than T with the mean of values at below T.

```
x = np.array([-2, -1, 0, 5, 10])
T = 5
```

```
# Step 1: Replace negatives with 0: x -> [0,0,0,5,10]
# Values ≤ T are [0,0,0,5], mean = (0+0+0+5)/4 = 1.25
# then values at or below T=5 are [0,0,0,5], mean=1.25, final x=[0,0,0,5,1.25].
```

```
import numpy as np

def condition_replace(x, T):
    # Step 1: Replace all negative values with 0
    x = np.where(x < 0, 0, x)

    # Step 2: Replace values greater than T with mean of values <= T
    mean_below_T = x[x <= T].mean()
    x = np.where(x > T, mean_below_T, x)

    return x

# Example input
x = np.array([-2, -1, 0, 5, 10])
T = 5

# Apply condition replacement
result = condition_replace(x, T)

# Print result
print(result)
```

```
[0.   0.   0.   5.   1.25]
```

Task 4- 2D Linear Transformations

- Objective: Apply a linear transformation to a set of 2D points using matrix multiplication.

Task Details:

Inputs:

- points: (N, 2) array of points.
- T: (2, 2) transformation matrix.

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

# A 90-degree rotation matrix is:
T = np.array([[0, -1],
              [1,  0]])

# points_transformed:
# [[ 0, -1],
#  [ 1,  0],
#  [ 1, -1]]
```

```
import numpy as np

def apply_transformation(points, T):
    # Apply linear transformation using matrix multiplication
    points_transformed = points @ T.T
    return points_transformed

# Example input
```

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

# A 90-degree rotation matrix
T = np.array([[0, -1],
              [1,  0]])

# Apply transformation
points_transformed = apply_transformation(points, T)

# Print result
print(points_transformed)
```

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