Overview

This project applies a simplified version of the **rearrange** operation from the **Einops** library, with support for tensor manipulation operations via a pattern-based syntax. The implementation is applicable to NumPy arrays and supports reshaping, transposition, axis splitting/merging, and repetition.

Features Implemented in it

- Pattern-based tensor manipulation using **Einstein-inspired notation**
- Support for **Numpy** arrays
- Operations include:
    - **Reshaping**
    - **Transposition**
    - **Axis Splitting((h w) c -> h w c)**
    - **Axis Merging(h w c -> (h w) c)**
    - **Axis Repetition(a1c -> a b c)**
    - **Batch Dimensions(… h w -> … (h w))**

Implementation

The implementation approach use 3 functions one for **Parsing the Pattern,** another for **Validation of the shape** of the **Numpy** input arrays and the 3rd one being the **Rearrangement** function that follows a three-step process-:

1. **Initial Reshape**: Splits combined dimensions
2. **Axis Mapping**: Tracks axis positions for transformation
3. **Output Transformation**: Handles merging, reordering, and repetition

Requirements

For the requirements to code **rearrange** function from scratch, we need to install 3 libraries:

1.**Regex(Regular Expression)**

2. **Numpy**

3. **Typing**

Approach Description

1.Pattern Parsing

The pattern string is split into input and output parts using the -> delimiter. Each part is decomposed into atomic components using regular expressions. Special handling for:

  - Use parentheses () for axis splitting/merging
  - **Ellipsis(...)** for batch dimensions
  - Regular axis names

2.Shape Validation

- Verifies compatibility between input tensor shape and pattern
- Checks provided axis lengths
- Infers missing axis lengths when possible
- Validates split axis dimensions

3.Tensor Transformation Pipeline

Step1: Initial Reshape

- Processes input axes to split any combined dimensions
- Handles batch dimensions (ellipsis) by preserving them
- Creates intermediate tensor with all axes separated

Step2: Axis Position Mapping

- Builds a dictionary mapping axis names to their positions
- Special handling for:
  - Batch dimensions (tracked as ...0, ...1, etc.)
  - Split axes (maps inner axes to positions)
  - Simple axes (direct position mapping)

Step3: Output Transformation

- Determines output shape and transpose order
- Handles:

    - Merging axes via reshape
    - Reordering via transpose
    - New axes (repetition) via reshape
- Performs final reshape to target shape

Limitations:

- Supports only **NumPy** arrays
- Absence of assistance for reduction activities
- Restricted ability to recover from errors

Potential Improvements:

- Support for tensors in **PyTorch/TensorFlow** should be included**.**
- Execute reduction operations (max, mean, etc.).
- Needs to be optimized for very large tensors
- Needs more complex pattern validation

Testing Approach

The implementation includes comprehensive test cases covering:-

- Basic reshape and transpose operations
- Edge cases (empty dimensions, size-1 dimensions)
- Complex patterns combining multiple operations
- Error conditions (invalid patterns, shape mismatches)

Optimization Strategies

Several key optimizations to minimize the number of operations and improve performance:-

Single-Pass Pattern Parsing

- Using effective regular expressions, the pattern string is parsed exactly once.
- With the carefully constructed regex pattern r'\(.*?\)|\.\.\.|\w+', we utilise re.findall() to:- Capture paranthesized groups as single units identifying **Ellipsis(…)** for batch dimensions and to extract simple axis names.

Combined Shape Validation and Inference

- A single pass through the input axes is used for both shape validation and axis length inference.
- We then check for dimension compatibility, the criteria for splitting axis as per size, and for missing axis lengths.
- The above processes avoid separate validation and inference passes.

Minimal Reshape Operations

- We check whether an actual reshape is required or not before performing it
- The initial reshape combines all the necessary operations required for split in one step

Smart Transposition

- In this step, we skip Transpose operations when the axes are already in desired order.
- Compare the current and the required transpose order before operating

Batch Dimension Preserving

Batch Dimensions are handled separately so as to-:

- Avoid unnecessary reshape/transpose operations
- Maintain the original order

Making the operations memory efficient

- When feasible, all changes are made in place.
- Unnecessary operations are avoided by chaining operations, reusing the current tensor variable, and performing operations in the most efficient order.

So overall, we implemented 4 optimization techniques:

Lazy Evaluation that performs operations when absolutely necessary and checks the criteria before executing them.

Operation fusion, which combines multiple logical operations into a single function call, thereby handling splitting and margining in the same reshape operation.

Early Validation avoids fast failing of invalid patterns and shapes so as to avoid their failing later.

Memory Locality maximizes **Cache Efficiency** by minimizing memory allocations between operations.

While keeping the same functionality and accuracy, these optimisations speed up and reduce the amount of memory used in the implementation compared to a simple method.