Useful Dimension and Shape Operations for Tensors

Broadcasting in PyTorch

Broadcasting is a powerful feature in PyTorch that allows you to perform element-wise operations on tensors with different shapes. It simplifies operations by automatically expanding the smaller tensor to match the shape of the larger tensor, making them compatible for element-wise operations. Broadcasting follows these rules:

1. If two tensors have a different number of dimensions, the smaller tensor is "broadcast" to match the larger tensor's dimensions by adding new dimensions with size 1.
2. Starting from the trailing dimensions, if the size of a dimension in one tensor is 1, and the other tensor has a size greater than 1 in that dimension, the smaller tensor is broadcast along that dimension.
3. If after broadcasting, the dimensions of both tensors are still not compatible, PyTorch raises an error.

Broadcasting simplifies code and improves performance by avoiding the need to create explicit copies of data. It's commonly used in operations like adding a scalar to a tensor or combining tensors with different shapes during arithmetic operations.

Here's a simple example of broadcasting in PyTorch:

$$\text{Tensor A} : \text{shape } (3, 1)$$
$$\text{Tensor B} : \text{shape } (3, 4)$$

When you add these two tensors (`C = A + B`), broadcasting automatically replicates `A` along the second dimension to match the shape of `B`, resulting in a valid addition:

$$\text{Tensor A} : \text{shape } (3, 4)$$
$$\text{Tensor B} : \text{shape } (3, 4)$$
$$\text{Tensor C} : \text{shape } (3, 4)$$

This allows for concise and efficient code when working with tensors of varying shapes. Below you will find a list of useful methods to utilize broadcasting:

1. `size()`: Returns the size of the tensor.
    - **Description**: Provides the dimensions of the tensor.
    - **Usage**: Useful for understanding the shape of a tensor.
2. `view(size)`: Returns a tensor with a different shape.
    - **Description**: Reshapes the tensor without changing data.
    - **Usage**: Used for changing tensor dimensions while keeping the data.
3. `reshape(size)`: Similar to `view`, but can change size and stride.
    - **Description**: Reshapes the tensor and can handle non-contiguous tensors.
    - **Usage**: When reshaping tensors and non-contiguous data is involved.
4. `squeeze()`: Removes dimensions with size 1.
    - **Description**: Eliminates dimensions with a single element.
    - **Usage**: Commonly used to simplify tensor shapes.
5. `unsqueeze(dim)`: Adds a dimension of size 1.
    - **Description**: Introduces a new dimension at the specified position.
    - **Usage**: Useful when dimensionality needs to be increased.
6. `permute(*dims)`: Reorders dimensions.
    - **Description**: Rearranges tensor dimensions as specified.
    - **Usage**: When you need to change the order of dimensions.
7. `transpose(dim0, dim1)`: Swaps two dimensions.
    - **Description**: Exchanges the positions of two specified dimensions.
    - **Usage**: Typically for linear algebra operations.
8. `expand(*sizes)`: Expands dimensions to a larger size.
    - **Description**: Enlarges dimensions without copying data.
    - **Usage**: When dealing with tensors of different sizes.