PyTorch Tensor Operations - Practical Exercise

Computer Engineering - Neural Networks Course

Introduction

In this practical exercise, you'll learn how to create, manipulate, and perform operations on tensors using PyTorch. Tensors are the fundamental data structure in deep learning frameworks, similar to arrays in NumPy but with additional capabilities for GPU acceleration and automatic differentiation.

What are Tensors?

A tensor is a generalization of vectors and matrices to potentially higher dimensions. In PyTorch:

- A 0-dimensional tensor is a scalar
- A 1-dimensional tensor is a vector
- A 2-dimensional tensor is a matrix
- A tensor with more dimensions represents more complex data structures

Learning Objectives

After completing this exercise, you will be able to:

- Create tensors using different methods and from various data sources
- Perform basic mathematical operations on tensors
- Understand tensor dimensionality and reshaping operations
- Apply statistical operations to tensors
- Use broadcasting for efficient operations
- Implement common tensor operations used in deep learning

Exercise Structure

This exercise is divided into five sections:

1. **Creating and Manipulating Tensors** - Learn different ways to create tensors and basic manipulations
2. **Basic Tensor Operations** - Perform arithmetic and matrix operations
3. **Tensor Operations for Machine Learning** - Explore broadcasting, indexing, and device management
4. **Hands-on Exercises** - Practice what you've learned by solving problems
5. **Visualization** - See tensor operations in action through visualization

Prerequisites

- Basic understanding of Python
- Elementary knowledge of linear algebra concepts (vectors, matrices)
- Familiarity with NumPy is helpful but not required

Setup Instructions

1. Ensure you have PyTorch installed. If not, follow these instructions:

```
pip install torch torchvision matplotlib numpy

```

2. Open a Jupyter notebook or Python environment of your choice
3. Copy the provided code into your environment
4. Run each cell and observe the outputs
5. Complete the exercises in the designated sections

Part 1: Creating and Manipulating Tensors

In this part, you'll learn how to:

- Create tensors from Python lists and NumPy arrays
- Use specialized tensor creation functions
- Reshape tensors and manipulate their dimensions

Key Functions

`torch.tensor()`	Create a tensor from data
`torch.zeros()`	Create a tensor filled with zeros
`torch.ones()`	Create a tensor filled with ones
`torch.rand()`	Create a tensor with random values
`torch.arange()`	Create a tensor with a range of values
`tensor.reshape()`	Change tensor dimensions
`tensor.view()`	Change tensor dimensions (shares memory)
`tensor.T`	Transpose a tensor

Part 2: Basic Tensor Operations

In this part, you'll learn how to:

- Perform element-wise operations (addition, subtraction, multiplication)
- Perform matrix operations (matrix multiplication, dot product)
- Apply statistical functions (sum, mean, standard deviation)

Mathematical Operations

`tensor1 + tensor2`	Element-wise addition
`torch.add(tensor1, tensor2)`	Element-wise addition (function form)
`tensor1 * tensor2`	Element-wise multiplication
`torch.matmul(tensor1, tensor2)`	Matrix multiplication
`tensor1 @ tensor2`	Matrix multiplication (operator form)
`torch.dot(vector1, vector2)`	Dot product (for 1D tensors)

Part 3: Tensor Operations for Machine Learning

In this part, you'll learn about:

- Broadcasting - performing operations between tensors of different shapes
- Indexing and slicing - accessing specific elements of tensors
- Device management - moving tensors between CPU and GPU

Broadcasting in PyTorch

Broadcasting allows PyTorch to perform operations on tensors of different shapes. The smaller tensor is "broadcast" across the larger tensor to make the shapes compatible.
Example: Adding a vector (1D tensor) to each row of a matrix (2D tensor)

```
matrix = torch.rand(3, 4)  # 3x4 matrix
vector = torch.rand(4)     # Vector of length 4
result = matrix + vector   # Vector is broadcast to each row
```

Part 4: Hands-on Exercises

In this section, you'll apply what you've learned by solving the following exercises:

1. Create a 3×3 identity matrix using PyTorch functions
2. Perform a matrix multiplication between a 2×3 and a 3×2 matrix
3. Calculate the cosine similarity between two vectors
4. Normalize a matrix along rows (each row should sum to 1)
5. Create a function that calculates the Euclidean distance between two points represented as tensors

Instructions:

- Replace the "TODO" comments in the code with your implementation
- Run your code to check if it produces the expected output
- If you're stuck, you can refer to the solutions provided in the "Bonus" section, but try to solve the problems on your own first

Exercise 3: Cosine Similarity

The cosine similarity between two vectors $\mathbf{a}$ and $\mathbf{b}$ is defined as:

$$\text{cosine\_similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} \tag{1}$$

Where $\mathbf{a} \cdot \mathbf{b}$ is the dot product, and $|\mathbf{a}|$ is the magnitude (Euclidean norm) of vector $\mathbf{a}$. In PyTorch, this can be computed using `torch.nn.functional.cosine_similarity()`.

Part 5: Visualization

In this section, you'll:

- Visualize a 2D tensor as a heatmap
- Apply operations to the tensor and visualize the results
- Understand how tensor operations can be visualized for better comprehension

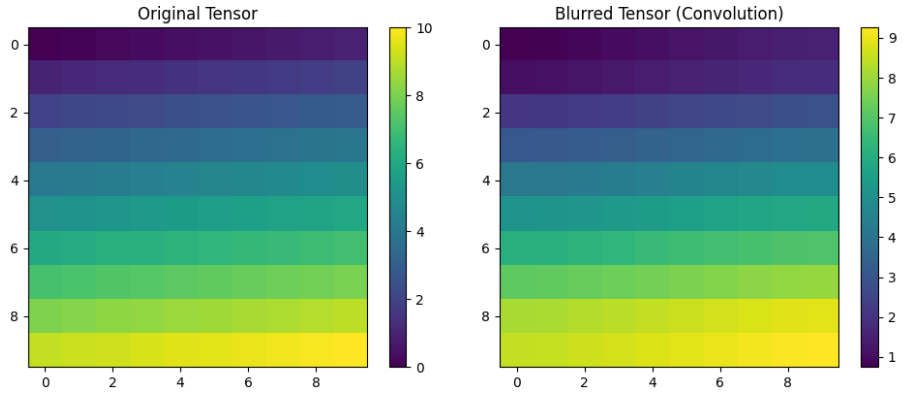

Figure 1: Example visualization of original and blurred tensor

Tips for Success

- Experiment beyond the provided examples by modifying the code
- Use PyTorch's documentation (https://pytorch.org/docs/stable/torch.html) as a reference
- Pay attention to tensor shapes when performing operations
- Note the similarities and differences between PyTorch tensors and NumPy arrays

Extension Activities

After completing the main exercises:

1. Try implementing a simple linear regression model using tensors and gradient computation
2. Experiment with more complex tensor operations like singular value decomposition or eigenvalue computation
3. Create your own tensor visualization that demonstrates an interesting mathematical concept

Submission Guidelines

Submit your completed notebook with all exercises solved. Include:

- All code with proper comments
- Explanations of your approach for each exercise
- Any interesting observations you made during the exercise
- Answers to the conceptual questions

Conceptual Questions

In addition to the coding exercises, answer the following questions:

1. How do PyTorch tensors differ from NumPy arrays? List at least three key differences.
2. Explain the concept of broadcasting in your own words and provide an example not shown in the lab.
3. What is the relationship between a tensor's shape and its dimensionality?
4. Why might you choose to use `tensor.view()` instead of `tensor.reshape()`?
5. Explain how PyTorch's automatic differentiation works with tensors.

Final Note

Tensor operations form the foundation of deep learning. By mastering these operations, you're building the skills needed to implement and understand more complex neural network architectures. Take your time with these exercises and make sure you understand each concept before moving forward.

Good luck, and enjoy exploring the world of tensor operations with PyTorch!