Lab 01 Answers Report

Hassan Almosa

Topic: *Artificial Intelligence COMP3742*
Lab Number: *1*
Campus: *Tonsely*

18 August 2025

Contents

1 What are the key features of PyTorch?

PyTorch is a **Python-first library** for tensor-based computation (like NumPy) with powerful **GPU acceleration** through CUDA integration. It allows building deep neural networks using a **tape-based autograd system**.
The main components of the PyTorch library are:

- `torch` – The core of PyTorch, responsible for tensor creation and extensive tensor operations (similar to NumPy, but with GPU acceleration).
- `torch.autograd` – A tape-based automatic differentiation library. It uses reverse-mode auto-differentiation, which is more dynamic compared to statically built neural network architectures in other frameworks. It records operations on the tape as you run them dynamically, resulting in less lag and overhead. Autograd supports all differentiable tensor operations in `torch`.
- `torch.jit` – A compilation stack (TorchScript) for creating serializable and optimizable models from PyTorch code.
- `torch.nn` – A module containing building blocks for neural networks, fully integrated with autograd.
- `torch.multiprocessing` – Python multiprocessing with concurrent access to shared tensors across processes.
- `torch.utils` – Data loading and utility module, providing tools for complex data loading, mapping, operations, and memory management.

2 How do you create a tensor in PyTorch?

```
tensor_a = torch.tensor([1, 2, 3])
tensor_b = torch.tensor([4, 5, 6])
```

3 What is the difference between element-wise multiplication and matrix multiplication in PyTorch?

- **Element-wise multiplication**: Each element in `tensor1` is multiplied by the corresponding element in `tensor2`. Both tensors must have the same shape. In PyTorch, this operation is denoted by `*`.
- **Matrix multiplication**: Follows the rules of linear algebra, where each entry is the sum of row–column products. This is crucial in neural networks, where each output neuron is a weighted sum of all inputs. In PyTorch, this operation is denoted by `@`.

Element-wise Example

```
A = torch.tensor([[1, 2],
                  [3, 4]])

B = torch.tensor([[5, 6],
                  [7, 8]])

C = A * B
print(C)
```

Output:

```
tensor([[ 5, 12],
        [21, 32]])
```

Matrix Multiplication Example

```
A = torch.tensor([[1, 2],
                  [3, 4]])

B = torch.tensor([[5, 6],
                  [7, 8]])

D = A @ B
print(D)
```

Output:

```
tensor([[19, 22],
        [43, 50]])
```

Step-by-step calculation:

- (1,1): $(1 \times 5) + (2 \times 7) = 5 + 14 = 19$
- (1,2): $(1 \times 6) + (2 \times 8) = 6 + 16 = 22$
- (2,1): $(3 \times 5) + (4 \times 7) = 15 + 28 = 43$
- (2,2): $(3 \times 6) + (4 \times 8) = 18 + 32 = 50$

4 How can you perform tensor slicing and indexing in PyTorch?

Given the tensors:

```
A = torch.tensor([[1, 2],
                  [3, 4]])

B = torch.tensor([[5, 6],
                  [7, 8]])
```

Indexing

```
print(A[0, 0])   # First row, first column -> 1
print(A[0, 1])   # First row, second column -> 2
print(A[1, 0])   # Second row, first column -> 3
print(A[1, 1])   # Second row, second column -> 4

# Negative indexing
print(A[-1])       # Last row -> tensor([3, 4])
print(A[:, -1])    # Last column -> tensor([2, 4])

# Boolean operations
print(A[A > 2])          # All elements greater than 2 -> tensor([3, 4])
print(B[B % 2 == 0])     # All even numbers in B -> tensor([6, 8])

# Indexing with lists
print(A[[0, 1], [1, 0]])  # Picks A[0,1] and A[1,0] -> tensor([2, 3])
```

Access

```
print(A[0])     # First row -> tensor([1, 2])
print(A[1])     # Second row -> tensor([3, 4])

print(A[:, 0])  # First column -> tensor([1, 3])
print(A[:, 1])  # Second column -> tensor([2, 4])
```

Slicing

```
print(A[0:2, 0:2])   # Whole matrix -> tensor([[1, 2], [3, 4]])
print(A[0:1, :])     # First row -> tensor([[1, 2]])
print(A[:, 0:1])     # First column (as column vector) -> tensor([[1], [3]])
```

5 Why is it recommended to use a virtual environment when installing PyTorch?

To avoid dependency and version conflicts, which are very common in Python libraries. Using a virtual environment ensures separation and isolation between projects, preventing countless conflicts and headaches.

References

[1] PyTorch Team. PyTorch: Open Source Machine Learning Library. https://github.com/pytorch/pytorch/tree/main?tab=readme-ov-file#more-about-pytorch

[2] PyTorch Team. PyTorch Documentation. https://docs.pytorch.org/docs/stable/index.html

[3] OpenAI. ChatGPT-5 Conversation Share Link 1.
https://chatgpt.com/share/68a2fea2-ae68-8011-bd4b-4b08097f9fb0

[4] OpenAI. ChatGPT-5 Conversation Share Link 2.
https://chatgpt.com/share/68a2e7cf-be8c-8011-9429-3f82502ca9a5

[5] OpenAI. ChatGPT-5 Conversation Share Link 3.
https://chatgpt.com/share/68a2febd-9d5c-8011-b855-b5ccc78892ca

[6] Apple. Core ML Documentation.
https://developer.apple.com/documentation/coreml

[7] Apple. MLX: Machine Learning Framework for Apple Silicon.
https://ml-explore.github.io/mlx/build/html/index.html

[8] Apple. Metal Backend for PyTorch.
https://developer.apple.com/metal/pytorch/

[9] Apple. Metal Performance Shaders Documentation.
https://developer.apple.com/documentation/metalperformanceshaders