LAB-2(Pytorch)

Explanation:

* PyTorch is an open-source machine learning library primarily used for deep learning applications.
* Developed by Facebook's AI Research lab (FAIR),
* it provides a flexible and easy-to-use platform for building and training neural networks,computer vision, natural language processing(electives in 6th sem)
* PyTorch is known for its dynamic computation graph(graph is built as operations are executed, making debugging and experimentation more intuitive, which allows developers to modify the network's structure on the fly, making it highly adaptable for research purposes.
* PyTorch uses tensors, which are multi-dimensional arrays similar to NumPy arrays, but with added support for GPU acceleration. This enables fast and efficient computations, especially for large-scale neural networks.

Installation: How to install PyTorch on your machine.

Creating Tensors: Various ways to create tensors in PyTorch.(tensor is a multi-dimensional array)

Tensor Operations: Basic operations that can be performed on tensors.

It is recommended, but not required, that your system has an NVIDIA GPU in order to harness the full power of PyTorch’s CUDA support. To use CUDA follow Nvidia's Installation Guide for Windows and Linux.

CUDA stands for **Compute Unified Device Architecture**. It's a parallel computing platform and application programming interface (API) created by NVIDIA. CUDA allows developers to use NVIDIA GPUs (Graphics Processing Units) for general-purpose processing, not just graphics rendering.

In simpler terms, CUDA lets your computer's graphics card do more than just make your video games look good. It helps with heavy computing tasks, like training deep learning models, by using the power of the GPU to do many calculations at once, much faster than a regular CPU could. This is why CUDA is widely used in scientific research, machine learning, and high-performance computing applications.

Tasks:

1. Obtain a tensor containing only zeros from the given tensor

import torch

pattern = torch.tensor([

[1, 1, 1, 1],

[1, 0, 0, 1],

[1, 0, 0, 1],

[1, 1, 1, 1]

])

zero\_tensor = torch.zeros\_like(pattern)

print(zero\_tensor)

2. Create a NumPy array of shape (3, 2, 3) using PyTorch

import torch

tensor = torch.rand(3, 2, 3)

numpy\_array = tensor.numpy()

print(numpy\_array.shape)

3. Create two random (2, 3, 3) tensors and find the max, min, mean, std of their product

import torch

tensor\_a = torch.rand(2, 3, 3)

tensor\_b = torch.rand(2, 3, 3)

product = torch.bmm(tensor\_a, tensor\_b)

max\_val = product.max()

min\_val = product.min()

mean\_val = product.mean()

std\_val = product.std()

print("Max:", max\_val)

print("Min:", min\_val)

print("Mean:", mean\_val)

print("Std:", std\_val)

4. Convert a 16x16 tensor into a 1x256 tensor

import torch

tensor\_16x16 = torch.rand(16, 16)

tensor\_1x256 = tensor\_16x16.view(1, -1)

print(tensor\_1x256.shape)

5. Find the coefficients that best model the linear relationship Y = ax + b (Linear Regression)

import torch

x = torch.tensor([1, 2, 3, 4, 5], dtype=torch.float32)

y = torch.tensor([2, 3, 5, 7, 11], dtype=torch.float32)

A = torch.vstack([x, torch.ones(len(x))]).T

coefficients, \_ = torch.lstsq(y, A)

a, b = coefficients[:2]

print("a:", a.item(), "b:", b.item())

6. Perform element-wise multiplication and addition on two 3x3 tensors

import torch

tensor\_a = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

tensor\_b = torch.tensor([[9, 8, 7], [6, 5, 4], [3, 2, 1]])

elementwise\_mul = tensor\_a \* tensor\_b

elementwise\_add = tensor\_a + tensor\_b

print("Element-wise Multiplication:")

print(elementwise\_mul)

print("Element-wise Addition:")

print(elementwise\_add)

7. Stack two 2x3 tensors along a new dimension

import torch

tensor\_a = torch.tensor([[1, 2, 3], [4, 5, 6]])

tensor\_b = torch.tensor([[7, 8, 9], [10, 11, 12]])

stacked\_tensor = torch.stack((tensor\_a, tensor\_b), dim=0)

print(stacked\_tensor)

8. Create a 1D tensor with values ranging from 0 to 9

import torch

tensor\_1d = torch.arange(0, 10)

print(tensor\_1d)

9. Perform operations on tensors of different shapes: 2x3 and 1x3 tensor using broadcasting

import torch

tensor\_a = torch.tensor([[1, 2, 3], [4, 5, 6]])

tensor\_b = torch.tensor([[10, 20, 30]])

result\_add = tensor\_a + tensor\_b

print("Result of Addition (A + B):")

print(result\_add)

10.Reshape a 1D tensor with 12 elements into a 3x4 matrix. give answer without comments

import torch

tensor\_1d = torch.arange(12)

tensor\_reshaped = tensor\_1d.view(3, 4)

print(tensor\_reshaped)