```
import torch
import numpy as np
# Tạo tensor từ list
data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)
print(f"Tensor từ list:\n {x_data}\n")
# Tạo tensor từ NumPy array
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
print(f"Tensor từ NumPy array:\n {x_np}\n")
# Tạo tensor với các giá trị ngẫu nhiên hoặc hằng số
x_ones = torch.ones_like(x_data) # tạo tensor gồm các số 1 có cùng shape với x_data
print(f"Ones Tensor:\n {x_ones}\n")
x_rand = torch.rand_like(x_data, dtype=torch.float) # tạo tensor ngẫu nhiên
print(f"Random Tensor:\n {x_rand}\n")
# In ra shape, dtype, và device của tensor
print(f"Shape của tensor: {x_rand.shape}")
print(f"Datatype của tensor: {x_rand.dtype}")
print(f"Device lưu trữ tensor: {x_rand.device}")
```

```
Tensor từ list:
 tensor([[1, 2],
        [3, 4]])

Tensor từ NumPy array:
 tensor([[1, 2],
        [3, 4]])

Ones Tensor:
 tensor([[1, 1],
        [1, 1]])

Random Tensor:
 tensor([[0.6604, 0.0876],
        [0.1627, 0.0565]])

Shape của tensor: torch.Size([2, 2])
Datatype của tensor: torch.float32
Device lưu trữ tensor: cpu
```

```
# 1. Cộng x_data với chính nó
x_sum = x_data + x_data
print(f"1. x_data + x_data:\n {x_sum}")

# 2. Nhân x_data với 5
x_multi = x_data * 5
print(f"2. x_data * 5:\n {x_multi}")

# 3. Nhân ma trận x_data với x_data.T
x_matrix_product = x_data @ x_data.T
print(f"3. x_data @ x_data.T (Nhân ma trận):\n {x_matrix_product}")
```

```
1. x_data + x_data:
 tensor([[2, 4],
        [6, 8]])
2. x_data * 5:
 tensor([[ 5, 10],
        [15, 20]])
3. x_data @ x_data.T (Nhân ma trận):
 tensor([[ 5, 11],
        [11, 25]])
```

```
# 1. Lấy ra hàng đầu tiên
row_1 = x_data[0, :]
print(f"1. Hàng đầu tiên (x_data[0, :]): {row_1}")

# 2. Lấy ra cột thứ hai
col_2 = x_data[:, 1]
print(f"2. Cột thứ hai (x_data[:, 1]): {col_2}")

# 3. Lấy ra giá trị ở hàng thứ hai, cột thứ hai
value_2_2 = x_data[1, 1]
print(f"3. Giá trị ở (Hàng 2, Cột 2) (x_data[1, 1]): {value_2_2}")
```

```
1. Hàng đầu tiên (x_data[0, :]): tensor([1, 2])
2. Cột thứ hai (x_data[:, 1]): tensor([2, 4])
```

```
3. Giá trị ở (Hàng 2, Cột 2) (x_data[1, 1]): 4
```

```
# Tạo tensor ngẫu nhiên với shape (4, 4)
x_4x4 = torch.rand(4, 4)
print(f"Tensor gốc (4x4):\n {x_4x4}\nShape: {x_4x4.shape}")

# Thay đổi hình dạng thành (16, 1) sử dụng view()
x_16x1_view = x_4x4.view(16, 1)
print(f"Tensor sau khi dùng view(16, 1):\n {x_16x1_view}\nShape: {x_16x1_view.shape}")
```

```
Tensor gốc (4x4):
 tensor([[0.6020, 0.2550, 0.5737, 0.7517],
        [0.0038, 0.8419, 0.5272, 0.9841],
        [0.6306, 0.3539, 0.4398, 0.5705],
        [0.7508, 0.6019, 0.9853, 0.4844]])
Shape: torch.Size([4, 4])
Tensor sau khi dùng view(16, 1):
 tensor([[0.6020],
        [0.2550],
        [0.5737],
        [0.7517],
        [0.0038],
        [0.8419],
        [0.5272],
        [0.9841],
        [0.6306],
        [0.3539],
        [0.4398],
        [0.5705],
        [0.7508],
        [0.6019],
        [0.9853],
        [0.4844]])
Shape: torch.Size([16, 1])
```

```
# Tạo một tensor và yêu cầu tính đạo hàm cho nó
x = torch.ones(1, requires_grad=True)
print(f"x: {x}")
# Thực hiện một phép toán
y = x + 2
print(f"y: {y}")
# y được tạo ra từ một phép toán có x, nên nó cũng có grad_fn
print(f"grad_fn của y: {y.grad_fn}")
# Thực hiện thêm các phép toán
z = y * y * 3
# Tính đạo hàm của z theo x
z.backward() # tương đương z.backward(torch.tensor(1.))
# Đạo hàm được lưu trong thuộc tính .grad
# Ta có z = 3 * (x+2)^2 => dz/dx = 6 * (x+2). Với x=1, dz/dx = 18
print(f"Đạo hàm của z theo x: {x.grad}")
```

```
x: tensor([1.], requires_grad=True)
y: tensor([3.], grad_fn=<AddBackward0>)
grad_fn của y: <AddBackward0 object at 0x7c7ca841c550>
Đạo hàm của z theo x: tensor([18.])
```

```
# Nếu gọi z.backward() một lần nữa, PyTorch sẽ gây ra lỗi runtime vì đã có gradient được tính toán và lưu trữ trong các lá Tens
# Lần 1 (z.backward()): PyTorch mở cuốn sổ, đọc tất cả các phép toán, tính toán gradient, và xé/vứt cuốn sổ đi (giải phóng bộ n
# Lần 2 (z.backward()): Bạn bảo PyTorch tính toán lại, nhưng cuốn sổ đã bị xé đi rồi. Nó không còn biết các phép toán $(+2)$, $
```

```
import torch
import torch.nn as nn

linear_layer = nn.Linear(in_features=5, out_features=2)
input_tensor = torch.randn(3, 5) # 3 mẫu, 5 chiều

output = linear_layer(input_tensor)

print(f"Input shape: {input_tensor.shape}")
# (3, 5) * (5, 2) -> (3, 2)
print(f"Output shape: {output.shape}")
print(f"Output:\n {output}\n")
```

```
Input shape: torch.Size([3, 5])
Output shape: torch.Size([3, 2])
Output:
 tensor([[ 0.4050,  0.0725],
```

```
        [-0.4264,  0.7796],
        [ 0.5765, -1.0622]], grad_fn=<AddmmBackward0>)
```

```
# Khởi tạo lớp Embedding cho một từ điển 10 từ, mỗi từ biểu diễn bằng vector 3 chiều
embedding_layer = torch.nn.Embedding(num_embeddings=10, embedding_dim=3)
# Tạo một tensor đầu vào chứa các chỉ số của từ (ví dụ: một câu)
# Các chỉ số phải nhỏ hơn 10
input_indices = torch.LongTensor([1, 5, 0, 8])
# Lấy ra các vector embedding tương ứng
embeddings = embedding_layer(input_indices)
print(f"Input shape: {input_indices.shape}")
print(f"Output shape: {embeddings.shape}")
print(f"Embeddings:\n {embeddings}")
```

```
Input shape: torch.Size([4])
Output shape: torch.Size([4, 3])
Embeddings:
 tensor([[ 0.5154,  1.1656, -0.4216],
        [ 3.3463,  0.3636, -1.1353],
        [-0.4016, -0.2516, -1.1566],
        [ 0.0390,  1.3621, -1.2428]], grad_fn=<EmbeddingBackward0>)
```

```
from torch import nn
class MyFirstModel(nn.Module):
  def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim):
    super(MyFirstModel, self).__init__()
    # Định nghĩa các lớp (layer) bạn sẽ dùng
    self.embedding = nn.Embedding(vocab_size, embedding_dim)
    self.linear = nn.Linear(embedding_dim, hidden_dim)
    self.activation = nn.ReLU() # Hàm kích hoạt
    self.output_layer = nn.Linear(hidden_dim, output_dim)
  def forward(self, indices):
    # Định nghĩa luồng dữ liệu đi qua các lớp
    # 1. Lấy embedding
    embeds = self.embedding(indices)
    # 2. Truyền qua lớp linear và hàm kích hoạt
    hidden = self.activation(self.linear(embeds))
    # 3. Truyền qua lớp output
    output = self.output_layer(hidden)
    return output
# Khởi tạo và kiểm tra mô hình
model = MyFirstModel(vocab_size=100, embedding_dim=16, hidden_dim=8, output_dim=2)
input_data = torch.LongTensor([[1, 2, 5, 9]]) # một câu gồm 4 từ
output_data = model(input_data)
print(f"Model output shape: {output_data.shape}")
```

```
Model output shape: torch.Size([1, 4, 2])
```