

Câu lệnh “!pip install torch”

```
[1] !pip install torch

Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.3.0+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.15.3)
Requirement already satisfied: typing-extensions<=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
Collecting nvidia-cuspars-cu12==12.1.0.106 (from torch)
  Using cached nvidia_cuspars-cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
Collecting nvidia-nccl-cu12==2.20.5 (from torch)
  Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl (176.2 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch)
  Using cached nvidia_nvtx-cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.3.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.3.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch)
  Downloading nvidia_nvjitlink_cu12-12.5.48-py3-none-manylinux2014_x86_64.whl (21.3 MB)
  21.3/21.3 MB 56.9 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath<1.4.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12-12.1.3.1, nvidia-cuda-cupti-cu12-12.1.105, nvidia-cuda-nvrtc-cu12-12.1.105, nvidia-cuda-runtime-cu12-12.1.105, nvidia-cudnn-cu12-8.9.2.26, nvidia-cufft-cu12-11.0.2.54
Successfully installed nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-runtime-cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 nvidia-cufft-cu12-11.0.2.54
0 giây hoàn thành lúc 01:05
```

Giải thích chi tiết : ở câu lệnh này, thì em tiến hành cài đặt thư viện Pytorch để thao tác hơn với GPU

Ô thực hiện đoạn mã thứ 2

```
import torch
import torch.nn.functional as F
embeddings = torch.tensor([[1.0, 0.0],
                             [0.0, 1.0],
                             [0.5, 0.5]])

W_Q = torch.eye(2)
W_K = torch.eye(2)
W_V = torch.eye(2)

Q = torch.matmul(embeddings, W_Q)
K = torch.matmul(embeddings, W_K)
V = torch.matmul(embeddings, W_V)

print("Q:", Q)
print("K:", K)
print("V:", V)"/>
```

```
import torch
import torch.nn.functional as F
embeddings = torch.tensor([[1.0, 0.0],
                             [0.0, 1.0],
                             [0.5, 0.5]])
W_Q = torch.eye(2)
W_K = torch.eye(2)
W_V = torch.eye(2)

Q = torch.matmul(embeddings, W_Q)
K = torch.matmul(embeddings, W_K)
V = torch.matmul(embeddings, W_V)

print("Q:", Q)
print("K:", K)
print("V:", V)
```

```
Q: tensor([[1.0000, 0.0000],
          [0.0000, 1.0000],
          [0.5000, 0.5000]])
K: tensor([[1.0000, 0.0000],
          [0.0000, 1.0000],
          [0.5000, 0.5000]])
V: tensor([[1.0000, 0.0000],
          [0.0000, 1.0000],
          [0.5000, 0.5000]])
```

- **Giải thích chi tiết:**

- “import torch
- import torch.nn.functional as F”
 - Ở trong đoạn code này, thì em import pytorch vào, để em dễ dàng sử dụng các hàm có sẵn như hàm kích hoạt, mất mát, và các hàm tiện ích khác.
- “embeddings = torch.tensor([[1.0, 0.0],
[0.0, 1.0],
[0.5, 0.5]])”
 - **embeddings**: Đây là một tensor có kích thước (3, 2), đại diện cho các embedding của ba từ khác nhau. Mỗi hàng trong tensor là một vector embedding cho một từ.
 - **[1.0, 0.0]**: Embedding cho từ đầu tiên (từ “The”).
 - **[0.0, 1.0]**: Embedding cho từ thứ hai (từ “Cat”).
 - **[0.5, 0.5]**: Embedding cho từ thứ ba (từ “Sat”).
- “W_Q = torch.eye(2)
W_K = torch.eye(2)
W_V = torch.eye(2)”
 - **W_Q, W_K, W_V**: Đây là các ma trận trọng số cho việc tạo các vector Query (Q), Key (K) và Value (V). Ở đây, các ma trận trọng số đều là ma trận đơn vị (identity matrix) có kích thước (2, 2). Ma trận đơn vị có giá trị 1 trên đường chéo chính và 0 ở các vị trí khác.
 - **torch.eye(2)**: Tạo một ma trận đơn vị kích thước 2x2.
- Q = torch.matmul(embeddings, W_Q)
K = torch.matmul(embeddings, W_K)
V = torch.matmul(embeddings, W_V)

- **Q, K, V**: Đây là các tensor kết quả của việc nhân các embedding với các ma trận trọng số tương ứng.
- **torch.matmul(embeddings, W_Q)**: Nhân ma trận **embeddings** với ma trận **W_Q** để tạo ra vector Query (Q). Tương tự cho Key (K) và Value (V).
- Cụ thể:
 - **Q = torch.matmul(embeddings, W_Q)**:
 - Nhân ma trận embedding với ma trận trọng số **W_Q**.
 - Vì **W_Q** là ma trận đơn vị, kết quả của phép nhân sẽ là chính các vector embedding ban đầu.
 - **K = torch.matmul(embeddings, W_K)**:
 - Nhân ma trận embedding với ma trận trọng số **W_K**.
 - Tương tự, vì **W_K** là ma trận đơn vị, kết quả sẽ là các vector embedding ban đầu.
 - **V = torch.matmul(embeddings, W_V)**:
 - Nhân ma trận embedding với ma trận trọng số **W_V**.
 - Tương tự, vì **W_V** là ma trận đơn vị, kết quả sẽ là các vector embedding ban đầu.
- `print("Q:", Q)`
`print("K:", K)`
`print("V:", V)`
 - **print("Q:", Q)**: In ra các vector Query (Q).
 - **print("K:", K)**: In ra các vector Key (K).
 - **print("V:", V)**: In ra các vector Value (V).
- Do các ma trận trọng số là ma trận đơn vị, kết quả của các phép nhân sẽ là chính các vector embedding ban đầu. Vì vậy, Q, K và V sẽ giống với **embeddings**.


Ô thực hiện đoạn mã thứ 3:

```

    dk = Q.size(-1)
    scores = torch.matmul(Q, K.transpose(-2, -1)) / torch.sqrt(torch.tensor(dk, dtype=torch.float32))

    print("Attention scores:", scores)

```

 Attention scores: tensor([[0.7071, 0.0000, 0.3536],
 [0.0000, 0.7071, 0.3536],
 [0.3536, 0.3536, 0.3536]])

Giải thích chi tiết:

- **Khai Báo Kích Thước Chiều**:
 - “`dk = Q.size(-1)`”
 - **Q.size(-1)**: Trả về kích thước của chiều cuối cùng của tensor Q.

- **Q** là một tensor có kích thước (số từ, chiều embedding). Trong trường hợp này, Q có kích thước (3, 2), do đó `Q.size(-1)` trả về 2.
- **dk** đại diện cho kích thước của các vector Q (hoặc K). Kích thước này được sử dụng để chuẩn hóa điểm attention.
- **Tính Điểm Attention (Scaled Dot-Product Attention)**
 - “`scores = torch.matmul(Q, K.transpose(-2, -1)) / torch.sqrt(torch.tensor(dk, dtype=torch.float32))`”
 - **Phép nhân ma trận:**
 - **`torch.matmul(Q, K.transpose(-2, -1))`:** Thực hiện phép nhân ma trận giữa Q và K^T (ma trận chuyển vị của K).
 - **`K.transpose(-2, -1)`:** Chuyển vị của K, đổi chiều từ (3, 2) thành (2, 3) để phù hợp với phép nhân ma trận.
 - Kết quả của phép nhân ma trận này sẽ là một tensor có kích thước (3, 3), biểu diễn điểm attention giữa mỗi cặp từ trong câu.
 - **Chuẩn hóa điểm attention:**
 - **`torch.sqrt(torch.tensor(dk, dtype=torch.float32))`:** Tính căn bậc hai của kích thước chiều cuối cùng của Q (hoặc K). Ở đây, **dk** là 2, do đó **`torch.sqrt(torch.tensor(dk, dtype=torch.float32))`** sẽ trả về $\sqrt{2}$.
 - Chia kết quả của phép nhân ma trận cho $\sqrt{2}$ giúp chuẩn hóa điểm attention, tránh việc các giá trị trở nên quá lớn khi kích thước chiều embedding tăng lên.
- In Kết Quả
 - “`print("Attention scores:", scores)`”
 - **`print("Attention scores:", scores)`:** In ra các điểm attention giữa các từ trong câu.

Ô thực hiện đoạn mã thứ 4:

```
[ ] attention_weights = F.softmax(scores, dim=-1)
    print("Attention weights:", attention_weights)

⇒ Attention weights: tensor([[0.4555, 0.2246, 0.3199],
                             [0.2246, 0.4555, 0.3199],
                             [0.3333, 0.3333, 0.3333]])
```

Giải thích chi tiết:

- **Áp Dụng Hàm Softmax**
 - “`attention_weights = F.softmax(scores, dim=-1)`”

- Hàm softmax:
 - `F.softmax(scores, dim=-1)`: Áp dụng hàm softmax lên tensor `scores` dọc theo chiều cuối cùng (`dim=-1`).
 - Hàm softmax chuyển đổi các điểm attention thành các xác suất, sao cho tổng các trọng số attention cho mỗi từ là 1.

Ô thực hiện đoạn mã thứ 5:

```
[5] attention_output = torch.matmul(attention_weights, V)
    print("Attention output:", attention_output)
```



```
↔ Attention output: tensor([[0.6155, 0.3845],
                             [0.3845, 0.6155],
                             [0.5000, 0.5000]])
```

Giải thích chi tiết:

- Tính Đầu Ra Attention
 - “`attention_output = torch.matmul(attention_weights, V)`”
 - Phép nhân ma trận:
 - `torch.matmul(attention_weights, V)`: Thực hiện phép nhân ma trận giữa các trọng số attention (`attention_weights`) và các vector giá trị (`values`).
 - Kết quả của phép nhân ma trận này là một tensor mới, biểu diễn đầu ra attention cho mỗi từ trong câu.