Nama : Rizka Rahmadina

NIM : 1103204115

Technical Report :

Reproduce_Codes_00_pytorch_fundamentals

```
# Import torch
import torch

# Create random tensor with shape (10, 5)
Z = torch.rand(size=(10, 5))

# Print the random tensor and its shape
print(Z)
print(Z.shape)
```

Kode ini mengimpor pustaka PyTorch agar dapat menggunakan fungsionalitas yang ada, termasuk pembuatan tensor yang membuat tensor acak Z dengan menggunakan fungsi torch.rand(). Tensor ini memiliki bentuk (shape) (10, 5), yang berarti memiliki 10 baris dan 5 kolom, dan nilainya dihasilkan secara acak antara 0 dan 1 lalu mencetak tensor acak Z dan bentuk (shape) nya ke layar. Ini memungkinkan melihat nilai-nilai dalam tensor dan memahami dimensi tensor tersebut.

Secara keseluruhan, kode tersebut membuat sebuah tensor acak dengan bentuk (shape) 10x5, kemudian mencetak nilai-nilai dalam tensor dan bentuk (shape) tensor tersebut. Nilai dari shape (10, 5) dapat diubah sesuai dengan dimensi yang dibutuhkan.

```
# Create another random tensor with shape (1, 5)
Y = torch.rand(size=(1, 5))

# Transpose the second tensor
Y_transposed = Y.t()

# Perform matrix multiplication
X = torch.matmul(Z, Y_transposed)
```

```
# Print the result and its shape
print(X)
print(X.shape)
```

Kode ini membuat tensor acak Y dengan bentuk (shape) (1, 5) yang berarti memiliki 1 baris dan 5 kolom sehingga menghasilkan tensor transposisi Y_transposed dari tensor Y. Transposisi mengubah baris menjadi kolom dan sebaliknya. Dalam hal ini, karena Y memiliki bentuk (1, 5), setelah transposisi, Y_transposed akan memiliki bentuk (5, 1). Baris ini melakukan perkalian matriks antara tensor Z dan tensor transposisi Y_transposed. Perkalian matriks membutuhkan jumlah kolom dalam tensor pertama sama dengan jumlah baris dalam tensor kedua. Hasilnya adalah tensor X. Baris ini mencetak hasil perkalian matriks (X) dan bentuk (shape)nya ke layar.

Jadi, secara keseluruhan, kode tersebut membuat dua tensor acak (Z dan Y), mentransposisi tensor Y, melakukan perkalian matriks antara Z dan Y_transposed, dan mencetak hasilnya beserta bentuknya.

```
# Set random seed for reproducibility on CPU
torch.manual_seed(0)

# Check if GPU is available and set random seed for reproducibility on GPU
if torch.cuda.is_available():
    torch.cuda.manual_seed(0)

# Create random tensor with shape (10, 5)
Z = torch.rand(size=(10, 5))

# Print the random tensor and its shape
print(X)
print(X.shape)

# Create another random tensor with shape (1, 5)
Y = torch.rand(size=(1, 5))


# Perform matrix multiplication
X = torch.matmul(Z, Y_transposed)

# Print the result and its shape
X, X.shape
```

Kode ini mengatur seed acak untuk generator angka acak pada CPU menggunakan torch.manual_seed(0). Mengatur seed ini memastikan hasil acak dapat direproduksi jika seed-nya sama. Baris ini memeriksa apakah GPU tersedia (torch.cuda.is_available()). Jika GPU tersedia, maka seed acak juga diatur untuk GPU menggunakan torch.cuda.manual_seed(0). Baris ini membuat tensor acak Z dengan bentuk (shape) (10, 5) dan mencoba mencetak tensor X, tetapi tampaknya ada kesalahan karena X belum didefinisikan sebelumnya. Seharusnya ini adalah Z, yang sebelumnya telah dibuat. Dengan demikian, perlu diperbaiki menjadi print(Z) dan print(Z.shape). Ini membuat tensor acak Y dengan bentuk (shape) (1, 5). Baris ini melakukan perkalian matriks antara Z dan tensor transposisi Y_transposed. Hasilnya disimpan dalam tensor X. Baris ini mencetak tensor hasil perkalian matriks (X) dan bentuk (shape)nya ke layar.

Jadi, secara keseluruhan, kode tersebut mengatur seed acak untuk reproduktibilitas, membuat dua tensor acak (Z dan Y), melakukan perkalian matriks antara keduanya, dan mencetak hasilnya beserta bentuknya.

```
# Set random seed on the GPU to a different value
torch.cuda.manual_seed(5678)
```

kode ini mengatur seed acak untuk GPU menggunakan torch.cuda.manual_seed(5678). Seed acak pada GPU diperlukan agar operasi acak pada GPU, seperti pembuatan tensor acak atau operasi acak lainnya, dapat direproduksi dengan hasil yang konsisten jika seed-nya sama. Jika kode sebelumnya memiliki pengaturan seed acak pada CPU dengan torch.manual_seed(0), maka pengaturan seed acak pada GPU memberikan kontrol yang serupa untuk operasi acak yang dilakukan pada GPU. Ini berguna jika Anda ingin hasil acak yang sama pada CPU dan GPU ketika bekerja dengan PyTorch.

Penting untuk diingat bahwa untuk reproduktibilitas penuh, kita perlu memastikan bahwa seed acak pada CPU dan GPU diatur secara konsisten dan sesuai kebutuhan aplikasi atau eksperimen Anda.

```
# Set random seed on the GPU to a different value
torch.cuda.manual_seed(5678)

# Check for access to GPU
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Device: {device}")

# Create two random tensors on GPU with the new seed and different size
tensor_A = torch.rand(size=(3, 4)).to(device)
tensor_B = torch.rand(size=(3, 4)).to(device)
tensor_A, tensor_B
```

Baris pertama mengatur seed acak untuk GPU menggunakan torch.cuda.manual_seed(5678). Ini memastikan bahwa operasi acak pada GPU dapat direproduksi dengan hasil yang konsisten jika seed-nya sama. Baris ini memeriksa apakah GPU tersedia (torch.cuda.is_available()). Jika GPU tersedia, variabel device diatur ke "cuda"; jika tidak, diatur ke "cpu". Pesan "Device: cuda" atau "Device: cpu" kemudian dicetak ke layar sesuai dengan hasil pemeriksaan. Baris ini membuat dua tensor acak (tensor_A dan tensor_B) dengan ukuran (shape) (3, 4) pada GPU. Operasi .to(device) digunakan untuk memastikan tensor dibuat pada perangkat yang sesuai (GPU atau CPU) sesuai dengan nilai variabel device. Baris ini akan mencetak tensor tensor_A dan tensor_B ke layar. Namun, jika kode ini dijalankan secara interaktif atau diakhir program, output tidak akan terlihat karena hasil dari dua tensor ini hanya ditampilkan tanpa menggunakan fungsi print().

Jadi, secara keseluruhan, kode tersebut mengatur seed acak pada GPU, memeriksa ketersediaan GPU, dan membuat dua tensor acak dengan seed dan ukuran yang berbeda pada GPU.

```
# Perform matmul on tensor_A and tensor_B
# tensor_C = torch.matmul(tensor_A, tensor_B) # won't work because of
shape error
tensor_C = torch.matmul(tensor_A, tensor_B.T)
tensor_C, tensor_C.shape
```

Kode ini digunakan untuk menghitung hasil dari perkalian matriks antara tensor_A dan transposisi dari tensor_B (tensor_B.T). Operasi ini memungkinkan karena bentuk (shape) tensor memenuhi aturan perkalian matriks, yaitu jumlah kolom

tensor_A harus sama dengan jumlah baris tensor_B (atau transposisi dari tensor_B), dan baris terakhir mencetak hasil perkalian matriks (tensor_C) dan bentuk (shape)nya ke layar.

Jadi, secara keseluruhan, kode tersebut melakukan operasi perkalian matriks yang valid antara tensor_A dan transposisi dari tensor_B dan mencetak hasilnya bersama dengan bentuk (shape) tensor_C.

```
# Find the maximum and minimum values of tensor_C
max_value = torch.max(tensor_C)
min_value = torch.min(tensor_C)

max_value, min_value
```

torch.max(tensor_C): Fungsi ini mengembalikan nilai maksimum dari tensor tensor_C. Ini mencari nilai maksimum di seluruh tensor tanpa memperdulikan posisi indeksnya.

torch.min(tensor_C): Fungsi ini mengembalikan nilai minimum dari tensor tensor_C. Ini mencari nilai minimum di seluruh tensor tanpa memperdulikan posisi indeksnya.

Baris terakhir mencetak nilai maksimum dan minimum ke layar. Jadi, secara keseluruhan, kode tersebut menghitung nilai maksimum dan minimum dari tensor tensor_C dan mencetak hasilnya ke layar.

```
# Find the indices of the maximum and minimum values of tensor_C
max_index = torch.argmax(tensor_C)
min_index = torch.argmin(tensor_C)

max_index, min_index
```

torch.argmax(tensor_C): Fungsi ini mengembalikan indeks dari nilai maksimum dalam tensor tensor_C. Indeks ini menyatakan posisi di mana nilai maksimum pertama kali ditemukan dalam tensor saat tensor diurutkan secara flat.

torch.argmin(tensor_C): Fungsi ini mengembalikan indeks dari nilai minimum dalam tensor tensor_C. Indeks ini menyatakan posisi di mana nilai minimum pertama kali ditemukan dalam tensor saat tensor diurutkan secara flat.

Baris terakhir mencetak indeks dari nilai maksimum dan minimum ke layar.

Jadi, secara keseluruhan, kode tersebut menghitung indeks (posisi) dari nilai maksimum dan minimum dalam tensor tensor_C dan mencetak hasilnya ke layar.

```
# Set seed to a different value
torch.manual_seed(42)

# Create a random tensor with a different size
tensor_F = torch.rand(size=(1, 1, 1, 8))

# Remove single dimensions
tensor_G = tensor_F.squeeze()

# Print out tensors
print(tensor_F, tensor_F.shape)
print(tensor_G, tensor_G.shape)
```

Ini mengatur seed acak menggunakan torch.manual_seed(42). Seed acak digunakan untuk menghasilkan angka acak dan memastikan hasil yang konsisten jika seed-nya sama. Baris ini membuat tensor acak tensor_F dengan bentuk (shape) (1, 1, 1, 8). Ini adalah tensor dengan dimensi yang berbeda dari tensor sebelumnya yang mungkin telah dibuat sebelumnya dalam kode atau notebook yang sama. Baris ini menghilangkan dimensi dengan ukuran 1 dari tensor tensor_F menggunakan fungsi squeeze(). Ini menghasilkan tensor baru tensor_G yang memiliki dimensi yang lebih sederhana. Baris ini mencetak tensor asli (tensor_F) dan tensor yang sudah diubah (tensor_G) beserta bentuk (shape) masing-masing. Ini memungkinkan Anda melihat perbedaan dalam bentuk tensor sebelum dan setelah menghilangkan dimensi yang berukuran 1.

Jadi, secara keseluruhan, kode tersebut mengatur seed acak, membuat tensor acak dengan ukuran (1, 1, 1, 8), menghilangkan dimensi dengan ukuran 1, dan mencetak hasilnya.

Inti dari semua kode yang telah disediakan dapat diringkas sebagai berikut:

1. Manipulasi Tensor dan Operasi Matriks:
    - Menciptakan tensor acak dengan berbagai bentuk menggunakan PyTorch (torch.rand).
    - Melakukan operasi matriks seperti perkalian matriks dan transposisi tensor.
2. Pengaturan Seed Acak:
    - Mengatur seed acak untuk memastikan reproduktibilitas hasil acak (torch.manual_seed dan torch.cuda.manual_seed).
3. Penggunaan GPU:
    - Mengecek ketersediaan GPU dan membuat tensor di GPU jika GPU tersedia (torch.cuda.is_available dan .to(device)).
4. Penemuan Nilai Maksimum dan Minimum:
    - Menggunakan torch.max dan torch.min untuk menemukan nilai maksimum dan minimum dari suatu tensor.
5. Penemuan Indeks Nilai Maksimum dan Minimum:
    - Menggunakan torch.argmax dan torch.argmin untuk menemukan indeks (posisi) dari nilai maksimum dan minimum dalam suatu tensor.
6. Manipulasi Dimensi Tensor:
    - Menggunakan squeeze() untuk menghilangkan dimensi dengan ukuran 1 dari suatu tensor.
7. Reproduktibilitas:
    - Pengaturan seed acak untuk memastikan hasil yang sama saat menggunakan angka acak.

Inti dari semua ini adalah pemahaman dan penerapan dasar-dasar operasi tensor, manipulasi dimensi, pengaturan seed acak, dan penggunaan GPU dalam lingkungan PyTorch.