Syifa Wanda Isnaini

1103201248

Machine Learning

00. Pytorch Fundamentals Exercise

```
# Import torch
import torch

# Create random tensor
X = torch.rand(size=(7, 7))
X, X.shape
```

Kode ini menggunakan PyTorch (torch) untuk membuat tensor acak dan mengeksplorasi propertinya. import torch Baris ini mengimpor pustaka PyTorch, yang digunakan untuk operasi tensor dan pembuatan model dalam konteks pembelajaran mesin. X = torch.rand(size=(7, 7)) Baris ini membuat tensor acak dengan menggunakan fungsi torch.rand(). Tensor ini memiliki ukuran (shape) 7x7. Fungsi torch.rand() menghasilkan tensor dengan nilai-nilai acak antara 0 dan 1 yang terdistribusi seragam. X, X.shape Baris ini mencetak tensor X dan shape-nya menggunakan properti shape. Properti ini memberikan informasi tentang ukuran (dimensi) dari tensor.

```
(tensor([[0.5656, 0.4012, 0.1987, 0.2464, 0.6861, 0.4953, 0.3433],
         [0.0032, 0.0228, 0.9020, 0.1267, 0.8009, 0.5274, 0.7453],
         [0.9123, 0.8138, 0.1667, 0.5998, 0.4657, 0.4473, 0.8367],
         [0.5302, 0.2213, 0.4747, 0.6485, 0.4770, 0.8675, 0.3054],
         [0.4226, 0.1398, 0.4495, 0.6974, 0.1808, 0.5872, 0.6931],
         [0.2153, 0.7517, 0.3505, 0.3815, 0.3244, 0.2511, 0.4269],
         [0.1158, 0.6696, 0.3733, 0.2633, 0.4102, 0.1101, 0.1613]]),
 torch.Size([7, 7]))
```

Output ini menunjukkan tensor X yang berisi nilai-nilai acak dan shape-nya yang merupakan ukuran 7x7. Isi dari tensor akan bervariasi setiap kali Anda menjalankan kode ini karena tensor dibuat dengan nilai-nilai acak.

```
# Create another random tensor
Y = torch.rand(size=(1, 7))
# Z = torch.matmul(X, Y) # will error because of shape issues
Z = torch.matmul(X, Y.T) # no error because of transpose
Z, Z.shape
```

```
(tensor([[1.0888],
         [1.7506],
         [1.8468],
         [1.7496],
         [1.9022],
         [1.2684],
         [0.8617]]), torch.Size([7, 1]))
```

Kode ini menciptakan dua tensor acak dan melakukan operasi perkalian matriks antara mereka menggunakan fungsi torch.matmul(). Y = torch.rand(size=(1, 7)) Baris ini membuat tensor acak Y dengan ukuran (shape) 1x7. Ini adalah tensor baris (row tensor) dengan satu baris dan tujuh kolom. Z = torch.matmul(X, Y.T) # no error because of transpose Baris ini memperbaiki masalah bentuk dengan mentranspos Y sebelum melakukan perkalian matriks. Operator .T digunakan untuk mentranspos tensor, yaitu menukar dimensi baris dan kolom. Dengan melakukan ini, kedua tensor memiliki dimensi yang sesuai untuk perkalian matriks. Z, Z.shape Baris ini mencetak hasil dari operasi perkalian matriks torch.matmul(X, Y.T) dan shape-nya.

Output ini menunjukkan tensor Z hasil dari perkalian matriks X dan Y.T (tensor Y yang sudah ditranspos). Hasilnya adalah tensor kolom dengan tujuh baris. Dalam operasi perkalian matriks, dimensi dari tensor hasil dipengaruhi oleh dimensi dari tensor-tensor yang dikalikan.

```
# Set manual seed
torch.manual_seed(0)

# Create two random tensors
X = torch.rand(size=(7, 7))
Y = torch.rand(size=(1, 7))

# Matrix multiply tensors
Z = torch.matmul(X, Y.T)
Z, Z.shape
```

```
(tensor([[1.8542],
         [1.9611],
         [2.2884],
         [3.0481],
         [1.7067],
         [2.5290],
         [1.7989]]), torch.Size([7, 1]))
```

Kode ini mengatur seed manual untuk memastikan reproduktibilitas hasil acak, kemudian membuat dua tensor acak dan melakukan operasi perkalian matriks di antara mereka. torch.manual_seed(0) Baris ini mengatur seed manual untuk generator bilangan acak PyTorch agar hasil acak dapat direproduksi secara konsisten. Dengan mengatur seed yang sama, hasil yang dihasilkan oleh fungsi-fungsi acak akan selalu sama pada setiap eksekusi program. X = torch.rand(size=(7, 7)) Y = torch.rand(size=(1, 7)) Baris ini membuat dua tensor acak, X dan Y. X adalah tensor 7x7 dengan nilai-nilai acak yang terdistribusi seragam antara 0 dan 1. Y adalah tensor baris 1x7 dengan nilai-nilai acak yang juga terdistribusi seragam antara 0 dan 1. Z = torch.matmul(X, Y.T) Baris ini melakukan operasi perkalian matriks antara X dan transpos dari Y (Y.T). Seperti yang dibahas sebelumnya, transpos Y diperlukan untuk memastikan dimensi yang sesuai untuk perkalian matriks. Z, Z.shape Baris ini mencetak hasil dari operasi perkalian matriks (Z) dan shape-nya. Output ini menunjukkan tensor hasil dan bentuknya setelah perkalian matriks.

Output ini menunjukkan tensor Z hasil dari perkalian matriks X dan Y.T (tensor Y yang sudah ditranspos). Hasilnya adalah tensor kolom dengan tujuh baris.

```
# Set random seed on the GPU
torch.cuda.manual_seed(1234)
```

Kode ini digunakan untuk mengatur seed acak pada GPU di PyTorch. torch.cuda.manual_seed(1234):

Fungsi ini digunakan untuk mengatur seed acak pada GPU di PyTorch. Saat kita menggunakan GPU untuk perhitungan, terutama dalam konteks pembelajaran mesin atau pelatihan model, terdapat operasi-operasi acak yang dapat mempengaruhi hasil. Sebagai contoh, inisialisasi bobot model pada layer-layer yang memiliki elemen acak, seperti dropout atau operasi-operasi lain yang melibatkan randomness. Dengan mengatur seed acak pada GPU menggunakan torch.cuda.manual_seed(1234), kita mencoba untuk memastikan bahwa hasil acak pada GPU akan tetap konsisten antar-eksekusi program, selama kondisi seed yang sama dipertahankan. Seed Value:

Angka 1234 adalah nilai seed yang dipilih. Pemilihan nilai seed ini bersifat sembarang, dan angka lain juga bisa digunakan. Penting untuk dicatat bahwa untuk mendapatkan hasil yang benar-benar konsisten, baik seed pada GPU (torch.cuda.manual_seed) maupun pada CPU (torch.manual_seed) harus diatur dengan nilai yang sama. Pengaturan seed acak ini berguna terutama ketika Anda ingin menjaga reproduktibilitas hasil eksperimen atau pelatihan model, sehingga setiap kali program dijalankan dengan seed yang sama, hasil yang dihasilkan oleh operasi-operasi acak pada GPU akan tetap konsisten.

```
# Set random seed
torch.manual_seed(1234)

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

# Create two random tensors on GPU
tensor_A = torch.rand(size=(2,3)).to(device)
tensor_B = torch.rand(size=(2,3)).to(device)
tensor_A, tensor_B
```

```
evice: cuda
(tensor([[0.0290, 0.4019, 0.2598],
         [0.3666, 0.0583, 0.7006]], device='cuda:0'),
 tensor([[0.0518, 0.4681, 0.6738],
         [0.3315, 0.7837, 0.5631]], device='cuda:0'))
```

Kode ini mengatur seed acak untuk hasil reproduktibilitas dan kemudian membuat dua tensor acak di PyTorch, memastikan bahwa tensor-tensor tersebut berada pada perangkat (device) GPU jika tersedia torch.manual_seed(1234): Fungsi ini digunakan untuk mengatur seed acak pada CPU di PyTorch. Hal ini memastikan bahwa operasi-operasi acak yang melibatkan CPU, seperti pembuatan tensor acak, akan menghasilkan nilai yang konsisten setiap kali program dijalankan dengan seed yang sama. device = "cuda" if torch.cuda.is_available() else "cpu" print(f"Device: {device}") Check for GPU Availability: Baris ini mengecek ketersediaan GPU menggunakan torch.cuda.is_available(). Jika GPU tersedia, variabel device diatur sebagai "cuda", menunjukkan bahwa GPU dapat digunakan. Jika tidak, variabel device diatur sebagai "cpu". tensor_A = torch.rand(size=(2, 3)).to(device) tensor_B = torch.rand(size=(2, 3)).to(device) Create Two Random Tensors on GPU: Baris ini membuat dua tensor acak (tensor_A dan tensor_B) dengan ukuran 2x3 dan memindahkan tensor-tensor tersebut ke perangkat (device) yang telah ditentukan (device) menggunakan .to(device).

Output ini menunjukkan bahwa tensor-tensor tensor_A dan tensor_B telah berhasil dibuat di GPU ("cuda:0"). Pemindahan ke GPU dilakukan menggunakan .to(device), dan informasi perangkat (device='cuda:0') ditampilkan dalam output.

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

```
(tensor([[0.3647, 0.4709],
         [0.5184, 0.5617]], device='cuda:0'), torch.Size([2, 2]))
```

Kode ini mencoba melakukan operasi perkalian matriks antara dua tensor (tensor_A dan tensor_B), dan kemudian mencetak hasilnya bersama dengan bentuk (shape) dari tensor hasil. tensor_C = torch.matmul(tensor_A, tensor_B.T) Baris ini mencoba melakukan operasi perkalian matriks antara tensor_A dan transpos dari tensor_B (tensor_B.T). Komentar yang diabaikan (# won't work because of shape error) menunjukkan bahwa sebelumnya mencoba melakukan operasi perkalian matriks langsung antara tensor_A dan tensor_B akan menghasilkan error karena masalah bentuk (shape). Oleh karena itu, transpos tensor_B diperlukan agar dimensi sesuai untuk perkalian matriks. tensor_C, tensor_C.shape : Baris ini mencetak hasil dari operasi perkalian matriks (tensor_C) dan shape-nya.

Output ini menunjukkan tensor hasil tensor_C dari operasi perkalian matriks torch.matmul(tensor_A, tensor_B.T). Tensor hasil memiliki shape [2, 2], yang sesuai dengan dimensi yang diharapkan dari operasi perkalian matriks tersebut.

```
# Find max
max = torch.max(tensor_C)

# Find min
min = torch.min(tensor_C)
max, min
```

```
(tensor(0.5617, device='cuda:0'), tensor(0.3647, device='cuda:0'))
```

(tensor(0.5617, device='cuda:0'), tensor(0.3647, device='cuda:0'))
Kode ini mencari nilai maksimum (max) dan nilai minimum (min) dari tensor tensor_C yang dihasilkan dari operasi perkalian matriks sebelumnya. torch.max(tensor_C):

Fungsi ini digunakan untuk mencari nilai maksimum dari seluruh elemen dalam tensor tensor_C. torch.min(tensor_C):

Fungsi ini digunakan untuk mencari nilai minimum dari seluruh elemen dalam tensor tensor_C. max, min: Baris ini mencetak nilai maksimum dan minimum yang telah ditemukan.

Output ini menunjukkan bahwa nilai maksimum dari tensor tensor_C adalah 1.1447, dan nilai minimumnya adalah 0.5392. Kedua nilai ini ditemukan menggunakan fungsi torch.max dan torch.min pada tensor hasil perkalian matriks.

```
# Find arg max
arg_max = torch.argmax(tensor_C)

# Find arg min
arg_min = torch.argmin(tensor_C)
arg_max, arg_min
```

```
(tensor(3, device='cuda:0'), tensor(0, device='cuda:0'))
```

Kode ini mencari indeks (posisi) dari nilai maksimum (arg_max) dan nilai minimum (arg_min) dalam tensor tensor_C. torch.argmax(tensor_C):

Fungsi ini digunakan untuk mencari indeks dari nilai maksimum dalam tensor tensor_C. Indeks yang ditemukan merupakan indeks yang sesuai dengan urutan flat (rata) dari tensor yang telah di-flatten. torch.argmin(tensor_C):

Fungsi ini digunakan untuk mencari indeks dari nilai minimum dalam tensor tensor_C. Indeks yang ditemukan merupakan indeks yang sesuai dengan urutan flat (rata) dari tensor yang telah di-flatten. arg_max, arg_min Baris ini mencetak indeks dari nilai maksimum dan minimum yang telah ditemukan.

Output ini menunjukkan bahwa indeks dari nilai maksimum dalam tensor tensor_C adalah 2, dan indeks dari nilai minimumnya adalah 1. Kedua indeks ini ditemukan menggunakan fungsi torch.argmax dan torch.argmin pada tensor hasil perkalian matriks.

```
# Set seed
torch.manual_seed(7)

# Create random tensor
tensor_D = torch.rand(size=(1, 1, 1, 10))

# Remove single dimensions
tensor_E = tensor_D.squeeze()

# Print out tensors
print(tensor_D, tensor_D.shape)
print(tensor_E, tensor_E.shape)
```

```
tensor([[[[0.5349, 0.1988, 0.6592, 0.6569, 0.2328, 0.4251, 0.2071, 0.6297,
           0.3653, 0.8513]]]]) torch.Size([1, 1, 1, 10])
tensor([0.5349, 0.1988, 0.6592, 0.6569, 0.2328, 0.4251, 0.2071, 0.6297, 0.3653,
        0.8513]) torch.Size([10])
```

Kode ini mengatur seed acak, membuat tensor acak dengan dimensi (shape) (1, 1, 1, 10), dan kemudian menghilangkan dimensi-dimensi yang bersifat singleton menggunakan fungsi squeeze(). torch.manual_seed(7): Mengatur seed acak pada CPU menggunakan fungsi torch.manual_seed(). Ini bertujuan untuk memastikan hasil acak yang reprodusibel dan konsisten. torch.rand(size=(1, 1, 1, 10)):

Membuat tensor acak tensor_D dengan dimensi (shape) (1, 1, 1, 10). Tensor ini memiliki satu dimensi yang bukan singleton (dimensi dengan ukuran lebih dari 1), yaitu dimensi terakhir (dimensi keempat) dengan ukuran 10. tensor_D.squeeze(): Menghilangkan dimensi-dimensi yang bersifat singleton dari tensor tensor_D. Dimensi yang dianggap singleton adalah dimensi yang memiliki ukuran 1. Sebagai contoh, jika sebuah tensor memiliki dimensi (1, 1, 1, 10), operasi squeeze() akan menghasilkan tensor dengan dimensi (10), yaitu menghilangkan semua dimensi yang bersifat singleton. print(tensor_D, tensor_D.shape) print(tensor_E, tensor_E.shape) Baris ini mencetak tensor tensor_D beserta shape-nya sebelum penghilangan dimensi singleton. Kemudian, mencetak tensor tensor_E beserta shape-nya setelah penghilangan dimensi singleton menggunakan squeeze().

Output ini menunjukkan tensor sebelum (tensor_D) dan setelah (tensor_E) penghilangan dimensi singleton. Hasil akhir, tensor_E, memiliki shape (10) setelah menghilangkan dimensi yang bersifat singleton dari tensor_D.