CH-2

1. 다음 중 PyTorch Tensor에 대한 설명으로 **옳지 않은** 것은?

1. Tensor는 다차원 배열 데이터를 표현하는 데 사용.
2. Tensor는 CPU 또는 GPU 메모리에서 생성될 수 있다.
3. Tensor는 NumPy 배열과 유사하지만, GPU 가속을 지원하지 않는다. ❤
4. Tensor는 다양한 데이터 타입(float, int, bool 등)을 가질 수 있다.
5. Tensor는 자동 미분(autograd) 기능을 지원하여 신경망 학습에 사용된다.

2. 다음 중 PyTorch Tensor의 속성(property)에 대한 설명으로 **옳지 않은** 것은?

1. `shape` 속성은 Tensor의 각 차원 크기를 나타내는 튜플이다.
2. `dtype` 속성은 Tensor에 저장된 데이터의 자료형(예: float32, int64)을 나타낸다.
3. `device` 속성은 Tensor가 저장된 장치(CPU 또는 GPU)를 나타낸다.
4. `grad` 속성은 Tensor의 기울기(gradient)를 저장하며, 항상 초기화되어 있다. ❤
5. `requires_grad` 속성은 Tensor에 대한 기울기 계산 여부를 나타내는 불리언 값이다.

3. 다음 중 PyTorch Tensor를 생성하는 방법으로 **올바르지 않은** 것은?

1. `torch.tensor()` 함수를 사용하여 Python 리스트나 NumPy 배열로부터 생성.
2. `torch.zeros()` 함수를 사용하여 0으로 채워진 Tensor를 생성.
3. `torch.ones()` 함수를 사용하여 1로 채워진 Tensor를 생성.
4. `torch.randn()` 함수를 사용하여 랜덤한 값을 가진 Tensor를 생성.
5. `torch.from_numpy()` 함수를 사용하여 NumPy 배열을 복사하여 새로운 Tensor를 생성. ❤ (NumPy 배열과 메모리를 공유하는 Tensor를 생성)

4. Python 코드에서 Tensor `z`와 `w`에 대한 설명으로 **옳지 않은** 것은?

```
import torch
```

```
x = torch.tensor([[1, 2], [3, 4]])
y = torch.tensor([[5, 6], [7, 8]])

z = torch.add(x, y)
w = torch.matmul(x, y)

print(z)
print(w)
```

1. z 는 x 와 y 의 요소별 합(element-wise addition) 결과를 담고 있다.
2. w 는 x 와 y 의 행렬 곱(matrix multiplication) 결과를 담고 있다.
3. torch.add() 함수는 Tensor의 요소별 덧셈을 수행한다.
4. torch.matmul() 함수는 Tensor의 요소별 곱셈을 수행한다. ❤
5. z 와 w 는 모두 2x2 크기의 Tensor이다.

5. 다음 Python 코드에서 Tensor element_wise_product 와 matrix_multiplication 에 대한 설명으로 **옳지 않은** 것은?

```
import torch

x = torch.tensor([[1, 2], [3, 4]])
y = torch.tensor([[5, 6], [7, 8]])

element_wise_product = torch.mul(x, y)
matrix_multiplication = torch.matmul(x, y)

print("Element-wise Product:")
print(element_wise_product)

print("\nMatrix Multiplication:")
print(matrix_multiplication)
```

```
Element-wise Product:
tensor([[ 5, 12], [21, 32]])
Matrix Multiplication:
tensor([[19, 22], [43, 50]])
```

1. element_wise_product 는 x 와 y 의 각 요소별 곱셈 결과를 담고 있다.
2. matrix_multiplication 은 x 와 y 의 행렬 곱셈 결과를 담고 있다.
3. torch.mul() 함수는 Tensor의 요소별 곱셈을 수행한다.
4. torch.matmul() 함수는 Tensor의 요소별 곱셈을 수행한다. ❤
5. 두 연산 모두 2x2 크기의 Tensor를 결과로 반환한다.

6. 다음의 4차원 PyTorch Tensor의 행렬 곱 예시 코드에 대하여 Tensor `x`와 `y`의 행렬 곱 연산 `torch.matmul(x, y)`에 대한 설명으로 **옳지 않은** 것은?

```
import torch

# 4차원 Tensor 생성 (batch_size, height, width, channel)
x = torch.randn(2, 3, 4, 5)
y = torch.randn(2, 3, 5, 6)

# 행렬 곱 수행
z = torch.matmul(x, y)

print("x shape:", x.shape)
print("y shape:", y.shape)
print("z shape:", z.shape)
```

```
x shape: torch.Size([2, 3, 4, 5])
y shape: torch.Size([2, 3, 5, 6])
z shape: torch.Size([2, 3, 4, 6])
```

1. `x`와 `y`의 마지막 두 차원(width, channel)에 대해 행렬 곱 연산이 수행된다.
2. `z`의 shape은 `(2, 3, 4, 6)`이다.
3. `x`의 마지막 차원 크기와 `y`의 마지막에서 두 번째 차원 크기는 같아야 한다.
4. `torch.matmul()` 함수는 4차원 Tensor에 대한 행렬 곱 연산을 지원하지 않는다. ❤
5. `z`의 각 요소는 `x`와 `y`의 해당 요소들의 행렬 곱 결과를 담고 있다.

7. PyTorch Tensor의 `detach()` 메서드에 대한 설명으로 **옳지 않은** 것은?

1. `detach()`는 Tensor의 복사본을 생성하지만, 계산 그래프에서 분리.
2. `detach()`된 Tensor는 기울기(gradient)를 추적하지 않는다.
3. `detach()`된 Tensor는 원본 Tensor와 메모리를 공유한다.
4. `detach()`는 Tensor의 `requires_grad` 속성을 `False`로 설정한니다.
5. `detach()`된 Tensor에서 수행된 연산은 원본 Tensor의 기울기에 영향을 미친다. ❤

8. 다음 Python 코드에서 그라디언트 계산에 대한 설명으로 **옳지 않은** 것은?

```
import torch

x = torch.tensor([1.0, 2.0, 3.0], requires_grad=True)
y = torch.tensor([4.0, 5.0, 6.0], requires_grad=True)
```

```
with torch.no_grad():
    z = x + y
    with torch.enable_grad():
        w = z * 2

print(w.requires_grad)
```

1. x와 y Tensor는 requires_grad=True로 설정되어 그라디언트 계산이 가능하다.
2. torch.no_grad() 컨텍스트 내의 연산은 그라디언트 계산 그래프에 기록되지 않는다.
3. torch.enable_grad() 컨텍스트는 torch.no_grad() 컨텍스트 내에서 특정 부분의 그라디언트 계산을 활성화한다.
4. w Tensor는 torch.enable_grad() 컨텍스트 내에서 생성되었으므로 requires_grad가 False이다. ❤
5. z Tensor는 torch.no_grad() 컨텍스트 내에서 생성되었으므로 그라디언트 계산 그래프에 포함되지 않는다.