PyTorch 스터디

23.08.01

강감찬

목차

1. 파이토치 패키지의 기본 구성

1) torch

- 메인 네임스페이스
- Numpy 와 유사한 구조
  - 배열 생성
  - 인덱싱 & 슬라이싱
  - 연산
  - 브로드캐스팅
  - 메모리 관리

1. 파이토치 패키지의 기본 구성

2) torch.autugrad

- 자동 미분(역전파) 위한 함수
- Context manager(객체 호출, 실행, 소멸 컨트롤)

1. 파이토치 패키지의 기본 구성

3) torch.nn

- 신경망 관련 데이터 구조, 레이어
- RNN, LSTM, ReLU, MSELoss

1. 파이토치 패키지의 기본 구성

4) torch.optim

- SGD 중심 파라미터 최적화 알고리즘

1. 파이토치 패키지의 기본 구성

5) torch.utils.data

- SGD 반복 연산 시 사용하는 미니 배치용 유틸 함수
- 전체 데이터 학습에는 많은 시간

  -> 데이터를 쪼개 학습

  > 쪼개는 단위: 미니 배치

1. 파이토치 패키지의 기본 구성

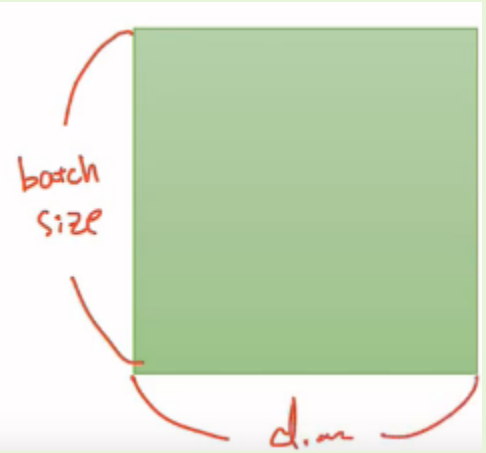
5) torch.onnx

- ONNX(Open Neural Network Exchange) 모델 export 시 사용
- 다른 프레임워크 간 모델 공유 시 사용하는 포맷

2. 텐서 조작하기 1

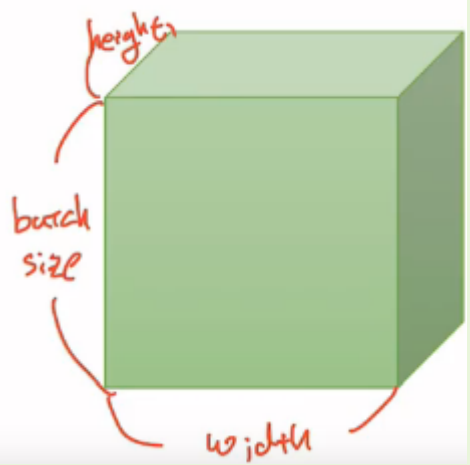
1) 2D Tensor(Matrix, Typical Simple Setting)

- batch size x dimension
- Batch size: 1 Epoch 시 처리하는 데이터 행 크기
- Dimension(차원): 행의 크기



2. 텐서 조작하기 1

2) 3D Tensor(Typical Computer Vision)



- 자연어 처리 예시
- Batch size x 문장 길이 x 단어 벡터 차원
- 단어: 특정 숫자의 집합
- 문장: 단어의 집합
- Batch: 문장의 집합

```
'나는' = [0.1, 0.2, 0.9]
'사과를' = [0.3, 0.5, 0.1]
'바나나를' = [0.3, 0.5, 0.2]
'좋아해' = [0.7, 0.6, 0.5]
'싫어해' = [0.5, 0.6, 0.7]
```

```
[['나는', '사과를', '좋아해'], ['나는', '바나나를', '좋아해'],
['나는', '사과를', '싫어해'], ['나는', '바나나를', '싫어해']]
```

```
[[[0.1, 0.2, 0.9], [0.3, 0.5, 0.1], [0.7, 0.6, 0.5]],
 [[0.1, 0.2, 0.9], [0.3, 0.5, 0.2], [0.7, 0.6, 0.5]],
 [[0.1, 0.2, 0.9], [0.3, 0.5, 0.1], [0.5, 0.6, 0.7]],
 [[0.1, 0.2, 0.9], [0.3, 0.5, 0.2], [0.5, 0.6, 0.7]]]
```

2. 텐서 조작하기 1

3) Numpy vs PyTorch: 기본 feature 비교

```
t_np = np.array([0, 1, 2, 3, 4, 5, 6])
print(f"t_np: {t_np}")
print(f"Rank of t_np: {t_np.ndim}")
print(f"Shape of t_np: {t_np.shape}₩n")

t_tensor = torch.IntTensor([0, 1, 2, 3, 4, 5, 6])
print(f"t_tensor: {t_tensor}")
print(f"Rank of t_tensor: {t_tensor.dim()}")
print(f"Shape of t_tensor: {t_tensor.shape}")
```

```
t_np: [0 1 2 3 4 5 6]
Rank of t_np: 1
Shape of t_np: (7,)

t_tensor: tensor([0, 1, 2, 3, 4, 5, 6], dtype=torch.int32)
Rank of t_tensor: 1
Shape of t_tensor: torch.Size([7])
```

```
t_2d_np = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10, 11, 12]
    ])
print(f"t_np: {t_2d_np}")
print(f"Rank of t_np: {t_2d_np.ndim}")
print(f"Shape of t_np: {t_2d_np.shape}₩n")

t_2d_torch = torch.IntTensor([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10, 11, 12]
    ])
print(f"t_torch: {t_2d_torch}")
print(f"Rank of t_torch: {t_2d_torch.dim()}")
print(f"Shape of t_torchr: {t_2d_torch.shape}")
```

```
t_np: [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
Rank of t_np: 2
Shape of t_np: (4, 3)

t_torch: tensor([[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9],
        [10, 11, 12]], dtype=torch.int32)
Rank of t_torch: 2
Shape of t_torchr: torch.Size([4, 3])
```

2. 텐서 조작하기 1

4) 브로드캐스팅

- 텐서 연산 시 데이터가 없는 차원 방향으로 데이터 복사 및 확장
- 정의되지 않은 차원으로 확장 X ex) 2D tensor -> 3D tensor
- https://colab.research.google.com/drive/1jr1wu6o2rQEwiiMZuWclYHWe-JX17tfx#scrollTo=_

2. 텐서 조작하기 1

5) 기본 연산 (원소 간 평균/덧셈/최대)

- dim argument로 방향 조절
- https://colab.research.google.com/drive/1jr1wu6o2rQEwiiMZuWclYHWe-JX17tfx#scrollTo=_

3. 텐서 조작하기 2

1) 뷰(view)

- 메모리를 재할당(realloc)

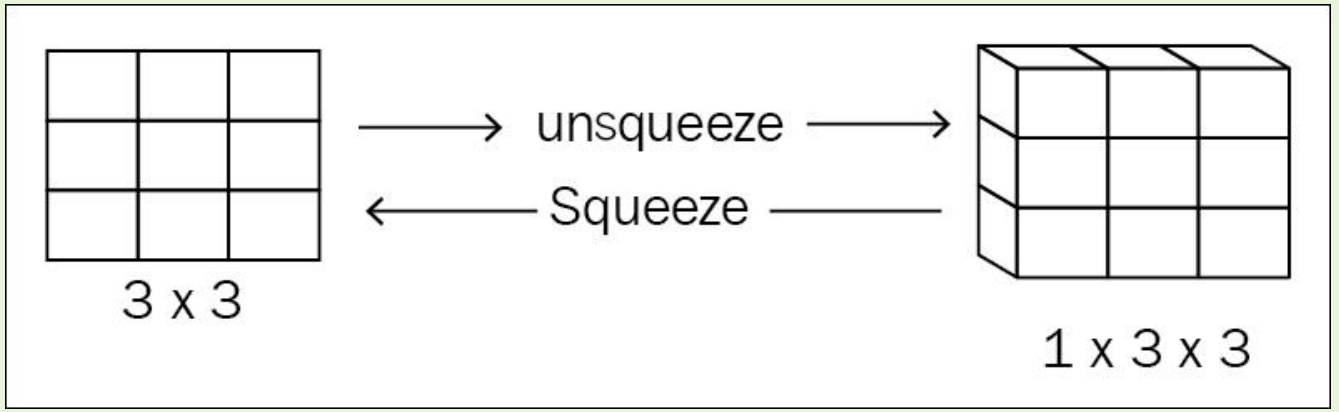
```
for i, add in enumerate(mem_address):
  if reshape_mem_address[i] != add:
    tmp = reshape_mem_address[i]
    print(f"index: {i}, ft address: {hex(add)}, reshape address: {hex(tmp)}")
  else:
    print(f"address: {hex(add)}")
```

```
address: 0x556d1543dc40
address: 0x556d1543dc44
address: 0x556d1543dc48
address: 0x556d1543dc4c
address: 0x556d1543dc50
address: 0x556d1543dc54
address: 0x556d1543dc58
address: 0x556d1543dc5c
address: 0x556d1543dc60
address: 0x556d1543dc64
address: 0x556d1543dc68
address: 0x556d1543dc6c
address: 0x556d1543dc70
address: 0x556d1543dc74
address: 0x556d1543dc78
address: 0x556d1543dc7c
address: 0x556d1543dc80
address: 0x556d1543dc84
address: 0x556d1543dc88
address: 0x556d1543dc8c
address: 0x556d1543dc90
address: 0x556d1543dc94
address: 0x556d1543dc98
address: 0x556d1543dc9c
```

3. 텐서 조작하기 2

2) Squeeze & Unsqueeze

- 길이 1인 차원 제거 & 추가
- 텐서 간 연산에서 브로드캐스팅 연산 용이



3. 텐서 조작하기 2

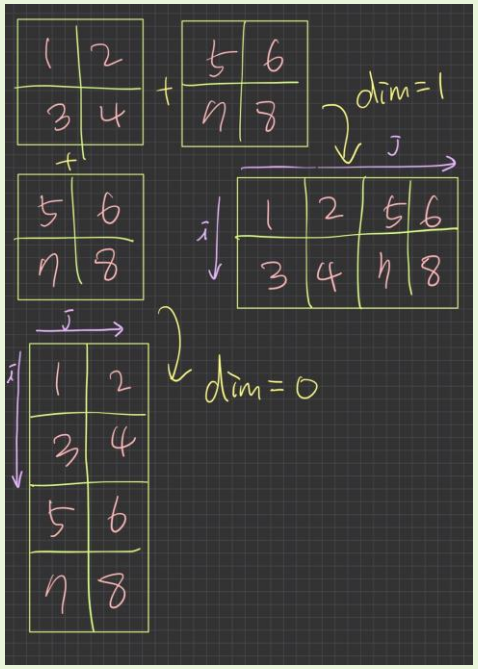
3) 타입 캐스팅

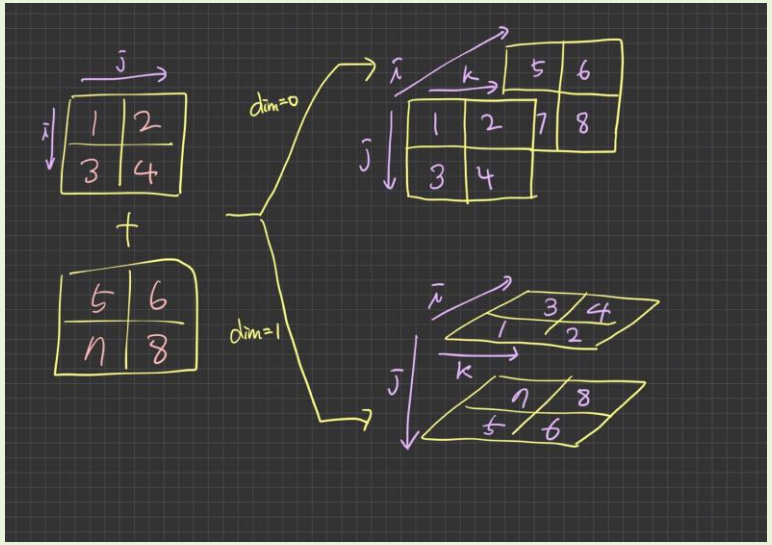
- 파이썬 -> 묵시적 형 변환(implicit type casting)이 자주 일어남
- 타입 지정이 중요 (data loss, 연산이 달라짐)

3. 텐서 조작하기 2

4) Concatenate & stack

- Concatenate: 일정 방향으로 텐서 연장
- Stack: 차원 추가 후 텐서 연장





3. 텐서 조작하기 2

4) ones_like & zeros_like

- 같은 크기의 모든 원소가 0 or 1로 된 텐서
- https://colab.research.google.com/drive/1jr1wu6o2rQEwiiMZuWclYHWe-JX17tfx#scrollTo=ones_like_zeros_like

4. 파이썬 클래스

- 캡슐화 (미지수, 변수 표현)
- 오버로딩 (특히 연산자)
- 동작에 집중 (연산 tree)
- 추상화 유리 (미분 연산)

출처

- p9, p10, :https://wikidocs.net/book/2788
- p15: https://subscription.packtpub.com/book/data/9781788834131/1/ch01lvl1sec06/getting-started-with-the-code

Q&A