PyTorch Basics

PyTorch = **numpy** + AutoGrad

- **Tensor**

  다차원 Arrays를 표현하는 PyTorch 클래스

  사실상 numpy의 ndarray와 동일

  tensor 생성하는함수도 거의 동일

  - numpy - ndarry

```
import numpy as np
n_array = np.arange(10).reshape(2,5)
print(n_array)
print("ndim:", n_array.ndim, "shape: ", n_array.shape)
```

  - pytorch - tensor

```
import torch
t_array = torch.FloatTensor(n_array)
print(t_array)
print("ndim:", t_array.ndim, "shape: ", t_array.shape)
```

  - data to tensor

```
data = [[3,5],[1,2]]
x_data = torch.tensor(data)
x_data
```

  - ndarray to tensor

```
nd_array_ex = np.array(data)
tensor_array = torch.from_numpy(nd_array_ex)
tensor_array
```

  - **tensor가 가질 수 있는 data 타입: 기본적으로 numpy와 동일**
  - pytorch의 대부분 사용법 그대로 적용

```
data = [[3,5,20],[10,5,50],[1,5,10]]
x_data = torch.tensor(data)
x_data[1:]
# tensor([[10,5,50],
#         [1,5,10]])
X_data[:2, 1:]
# tensor([[ 5, 20],
#         [ 5, 50]])
```

```
x_data.flatten()
# tensor([ 3, 5, 20, 10, 5, 50, 1, 5, 10])
torch.ones_like(x_data)
# tensor([[1, 1, 1],
#         [1, 1, 1],
#         [1, 1, 1]])
x_data.numpy()
# array([[ 3, 5, 20],
#        [10, 5, 50],
#        [ 1, 5, 10]], dtype=int64)
x_data.shape # torch.Size([3, 3])
x_data.dtype # torch.int64
```

- tensor는 GPU에 올려서 사용 가능

```
x_data.device # device(type='cpu')

if torch.cuda.is_available():
  x_data_cuda = x_data.to('cuda')
x_data_cuda.device
# device(type='cuda', index=0)
```

- **view**

  reshape과 동일, tensor의 shape 변환

  contiguity 보장

  reshape: contiguity 보장 X

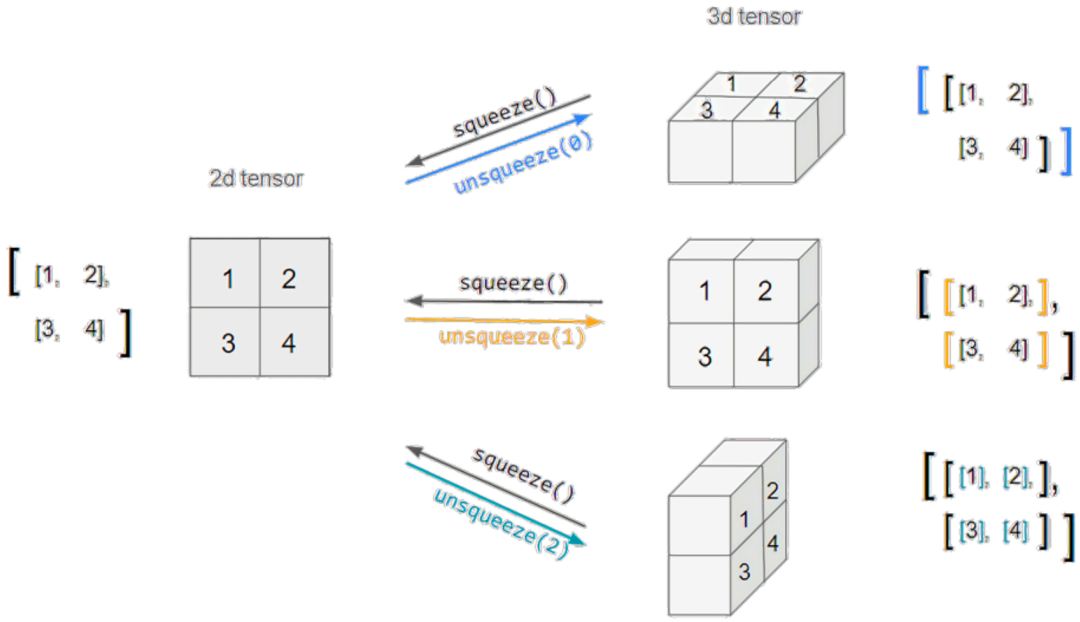
- **squeeze**

  차원의 개수가 1인 차원 삭제 (압축)

- **unsqueeze**

  차원의 개수가 1인 차원 추가

```
tensor_ex = torch.rand(size=(2, 3, 2)) tensor_ex
# tensor([[[0.7466, 0.5440],
#          [0.7145, 0.2119],
#          [0.8279, 0.0697]],
#          [[0.8323, 0.2671],
#          [0.2484, 0.8983],
#          [0.3228, 0.2254]]])
tensor_ex.view([-1, 6])
# tensor([[0.7466, 0.5440, 0.7145, 0.2119, 0.8279, 0.0697],
#         [0.8323, 0.2671, 0.2484, 0.8983, 0.3228, 0.2254]])
tensor_ex.reshape([-1,6])
# tensor([[0.7466, 0.5440, 0.7145, 0.2119, 0.8279, 0.0697],
#         [0.8323, 0.2671, 0.2484, 0.8983, 0.3228, 0.2254]])
```



- **operations도 numpy와 동일**
- 행렬곱셈 연산 함수는 dot 아니라 mm 사용
  - **mm: broadcasting 지원 X**
  - **matmul: broadcasting 지원**
- nn.functional: 다양한 수식 변환 지원

```
import torch
import torch.nn.functional as F

tensor = torch.FloatTensor([0.5, 0.7, 0.1])
h_tensor = F.softmax(tensor, dim=0)
h_tensor
# tensor([0.3458, 0.4224, 0.2318])

y = torch.randint(5, (10,5))
y_label = y.argmax(dim=1)
torch.nn.functional.one_hot(y_label)
```

- **AutoGrad**

  자동 미분 지원 => **backward 함수 사용**

  $y = w^2$

  $z = 10 * y + 25$

  $z = 10 * w^2 + 25$

```
w = torch.tensor(2.0, requires_grad=True)
y = w**2
z = 10*y + 25
z.backward()
w.grad
# tensor(40.)
```

$Q = 3a^3 - b^2$

$\frac{\sigma Q}{\sigma a} = 9a^2$

$\frac{\sigma Q}{\sigma b} = -2b$

```
a = torch.tensor([2., 3.], requires_grad=True)
b = torch.tensor([6., 4.], requires_grad=True)
Q = 3*a**3 - b**2
external_grad = torch.tensor([1., 1.]) Q.backward(gradient=external_grad)
a.grad
# tensor([36., 81.])
b.grad
# tensor([-12., -8.])
```