PyTorch Ch3

파이토치로 구현하는 ANN

채정아

INDEX

텐서 자유자재로 다루기

```python
import torch

x = torch.tensor([[1,1,1], [2,2,2], [3,3,3]])
print(x)
print("shape = ", x.shape)
print("차원 = ", x.ndimension())

x = torch.unsqueeze(x, 0) # 차원 늘리기
print(x)
print("shape = ", x.shape)
print("차원 = ", x.ndimension())
```

```
tensor([[1, 1, 1],
        [2, 2, 2],
        [3, 3, 3]])
shape =  torch.Size([3, 3])
차원 =  2
tensor([[[1, 1, 1],
         [2, 2, 2],
         [3, 3, 3]]])
shape =  torch.Size([1, 3, 3])
차원 =  3
```

```python
x = x.view(9)
print(x)
print("shape = ", x.shape)
print("차원 = ", x.ndimension())
```

```
tensor([1, 1, 1, 2, 2, 2, 3, 3, 3])
shape =  torch.Size([9])
차원 =  1
```

```python
w = torch.randn(4,2, dtype=torch.float)
y = torch.tensor([[3.0,3.0],[2.0,2.0]])
print(w)
print(y)
wy = torch.mm(w,y)    # 행렬 곱 연산
print(wy)
```

```
tensor([[ 1.6414,  1.5305],
        [ 0.9263,  0.6656],
        [ 1.1740, -0.9942],
        [ 0.0144,  1.3457]])
tensor([[3., 3.],
        [2., 2.]])
tensor([[7.9850, 7.9850],
        [4.1102, 4.1102],
        [1.5335, 1.5335],
        [2.7347, 2.7347]])
```

경사하강법으로 이미지 복원하기

- 이미지 처리를 위해 만들어 두었던 weird_function() 함수에 실수로 버그가 생겨 오염된 이미지가 만들어짐
- 원본 이미지로 복원하기 위해 경사하강법을 이용하자
- 해결 과정

1. 오염된 이미지와 같은 크기의 random_tensor 생성
2. 랜덤 텐서를 weird_function() 에 입력해 똑같이 오염된 이미를 가설hypothesis 라고 부르자
   (a) 원본 이미지가 weird_function()에 입력되어 오염된 이미지 출력
   (b) 인위적으로 생성한 이미지가 weird_function()에 입력되어 가설을 출력
3. 가설과 오염된 이미지가 같다면, 무작위 이미지 = 원본 이미지
4. weird_function(random_tensor) = broken_image 로 만들자

경사하강법으로 이미지 복원하기

```
import PIL.Image as pilimg
import torch
import matplotlib.pyplot as plt
import numpy as np

image = pilimg.open('super.png')
image = image.resize((100,100))      #픽셀 수 줄이기
pix = np.array(image)
new_image = torch.from_numpy(pix)    #numpy to torch

plt.imshow(new_image.view(100,100))      #원본
new_image = new_image.view(10000)
```

```
def weird_function(x, n_iter = 5):
    h = x
    filt = torch.tensor([-1./3, 1./3, -1./3])
    for i in range(n_iter):
        zero_tensor = torch.tensor([1.0*0])
        h_l = torch.cat((zero_tensor, h[:-1]), 0)
        h_r = torch.cat((h[1:], zero_tensor), 0)
        h = filt[0]*h + filt[2] * h_l + filt[1] * h_r
        if i%2 == 0:
            h = torch.cat( (h[h.shape[0]//2:], h[:h.shape[0]//2]), 0)
    return h

broken_image = weird_function(new_image.float())    #오염 시키기
plt.imshow(broken_image.view(100,100))
```

새로운 이미지로 실습하기 위해 이미지를 오염 시켜야 한다.

원본 사진을 Weird_function()에 입력해 broken_image를 얻었다.

경사하강법으로 이미지 복원하기

```
def distance_loss(hypothesis, broken_image):
    return torch.dist(hypothesis, broken_image)

random_tensor = torch.randn(10000, dtype = torch.float)
lr = 20      #learning late = 20
```

가설(무작위 이미지)와 broken_image 사이의 오차를 반환

학습률learning rate 을 20로 했을 때 가장 좋은 결과

```
for i in range(0,20000):
    random_tensor.requires_grad_(True)
    hypothesis = weird_function(random_tensor)
    loss = distance_loss(hypothesis, broken_image)
    loss.backward()
    with torch.no_grad():
        random_tensor = random_tensor - lr*random_tensor.grad
    if i % 1000 == 0:
        print('Loss at {} = {}'.format(i, loss.item()))


plt.imshow(random_tensor.view(100,100).data)
```

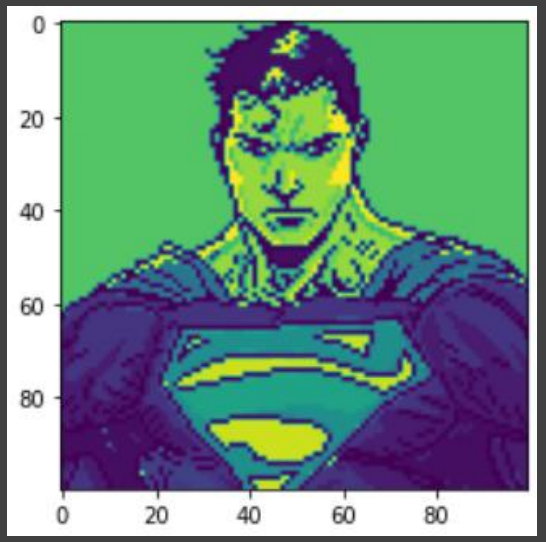
20000번 실행

랜덤 텐서를 weird_funtion에 입력해 가설 구함

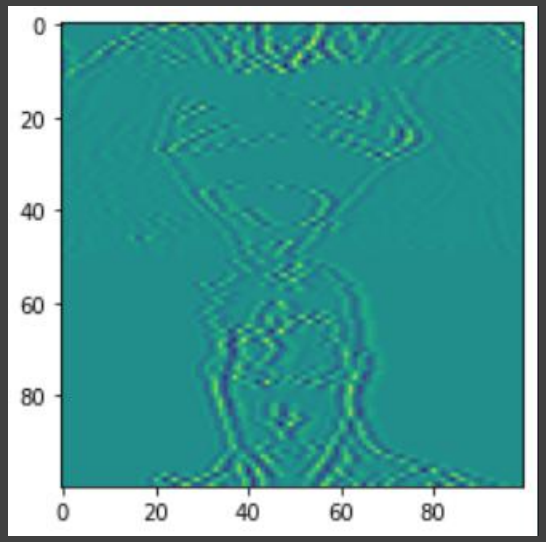
Random_tensor.grad => loss.backward() 에서 계산한 loss의 기울기, loss가 최댓점이 되는 곳의 방향

이 방향의 반대로 lr 만큼 이동

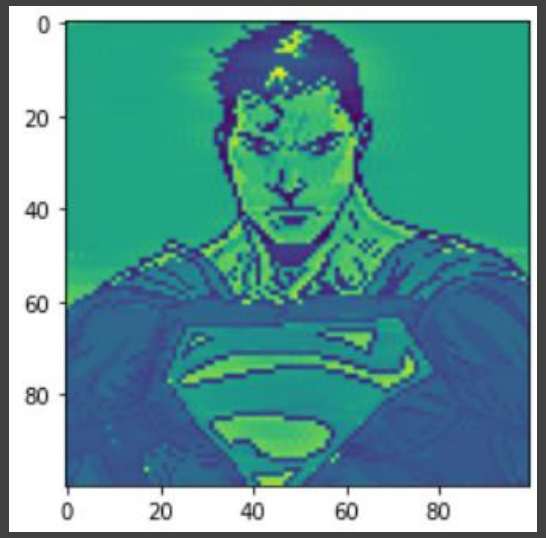
경사하강법으로 이미지 복원하기



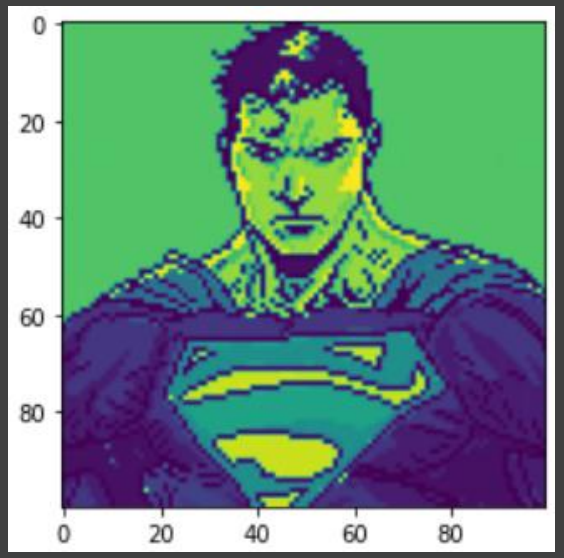
원본



broken_image



lr = 10



lr = 20

(가장 오차 작음)

```
Loss at 0 = 62.03315734863281
Loss at 1000 = 5.9893646240234375
Loss at 2000 = 5.3850836753845215
Loss at 3000 = 4.958432197570801
Loss at 4000 = 4.572556495666504
Loss at 5000 = 4.20366096496582
Loss at 6000 = 3.843784809112549
Loss at 7000 = 3.4893369674682617
Loss at 8000 = 3.1384098529815674
Loss at 9000 = 2.7899062633514404
Loss at 10000 = 2.443157434463501
Loss at 11000 = 2.0977349281311035
Loss at 12000 = 1.753369688987732
Loss at 13000 = 1.4098671674728394
Loss at 14000 = 1.06710946559906
Loss at 15000 = 0.7250255942344666
Loss at 16000 = 0.5288407206535339
Loss at 17000 = 0.5291553139686584
Loss at 18000 = 0.52919602394104
Loss at 19000 = 0.5291959047317505
```

감사합니다