exam 개인학습

Lec 5

Type something...

01) Multivariable Linear Regression

01) Multivariable Linear Regression

Hypothesis Function

$$H(x) = Wx + b$$

x 라는 vector 와 W 라는 matrix의 곱

$$H(x) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

입력변수가 3개라면 weight 도 3개!

- Multi linear 의 경우 Wx + b로 간단하게 행렬곱을 통해 나타낼 수 있다.

```
# 데이터
x_train = torch.FloatTensor([[73, 80, 75],
                             [93, 88, 93],
                             [89, 91, 90],
                             [96, 98, 100],
                             [73, 66, 70]])
y_train = torch.FloatTensor([[152], [185], [180], [196], [142]])

# 모델 초기화
W = torch.zeros((3, 1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)

# optimizer 설정
optimizer = optim.SGD([W, b], lr=1e-5)
```

1. 데이터 정의

2. 모델 정의

3. optimizer 정의

Full Code with torch.optim (2)

```
nb_epochs = 20
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = x_train.matmul(W) + b # or .mm or @

    # cost 계산
    cost = torch.mean((hypothesis - y_train) ** 2)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    print('Epoch {:4d}/{} hypothesis: {} Cost: {:.6f}'.format(
        epoch, nb_epochs, hypothesis.squeeze().detach(),
        cost.item()
    ))
```

4. Hypothesis 계산

5. Cost 계산 (MSE)

6. Gradient descent

- x_train의 dim = 3, **각각의 dim에다가 weight를 부여해줄 것이므로** `w.shape` = (3, 1) → `w = torch.zeros((3,1), require_grad= True)` → `x.matmul(w)` 로 계산
- 나머지는 Linear과 똑같은 방식으로 진행된다.

`hypothesis = x_train.matmul(w) + b`

`cost = torch.mean((hypothesis - y_train) ** 2) MSE`

`optimizer.zero_grad()`

`cost.backward()`

`optimizer.step()`