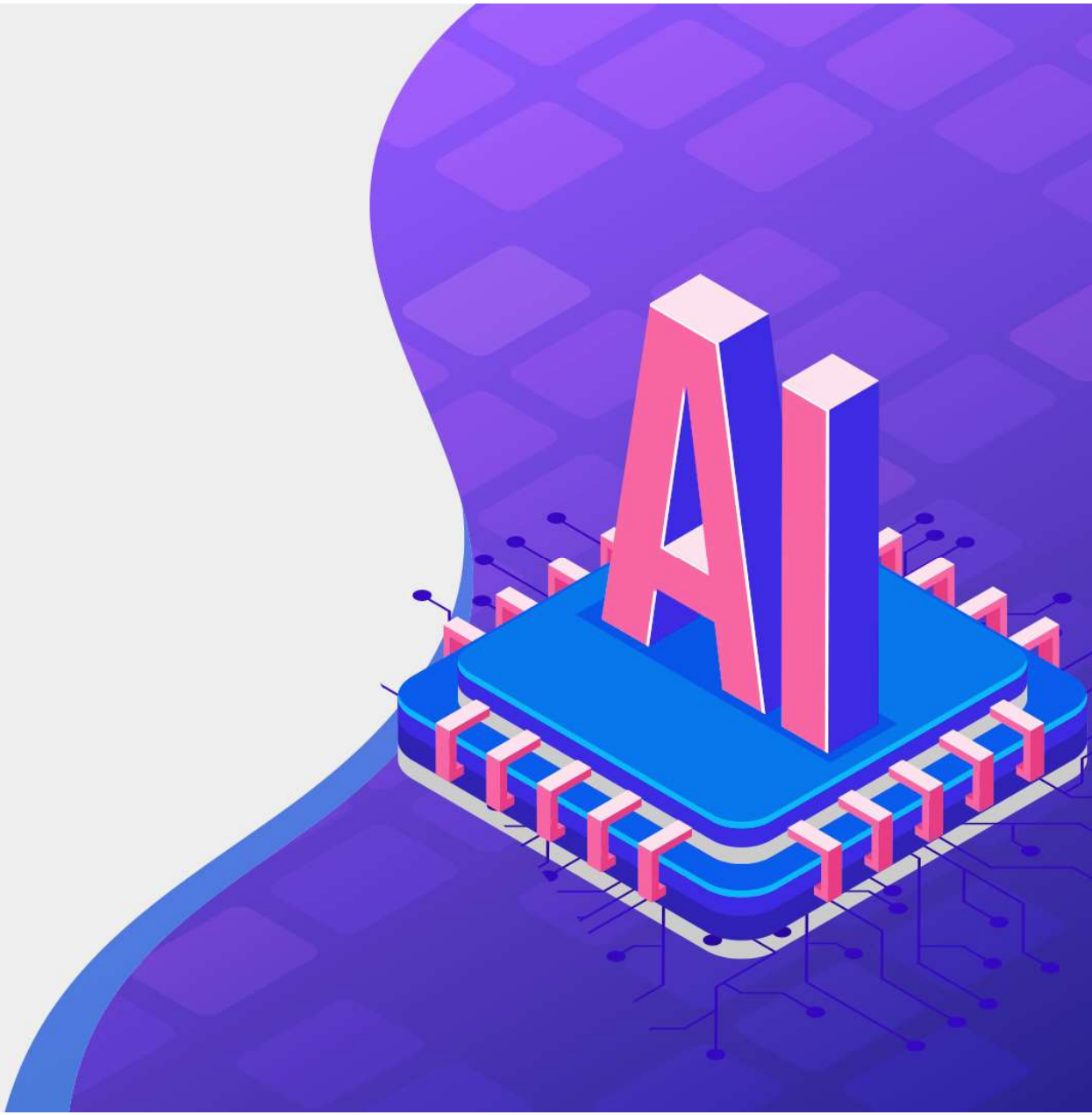

Linear Regression
(with Pytorch)
김재광 교수 (소프트웨어융합대학 글로벌융합학부)
AI

04 Linear Regression - with Pytorch

Contents

- Data definition
- Hypothesis
- Compute loss
- Gradient descent



Data definition

- What would be the grade if I study 4 hours?

4 hours →  → What point?

Hours	Points
1	2
2	4
3	6
4	??

Data definition (Cont'd)

```
x_train = torch.FloatTensor([1], [2], [3])
y_train = torch.FloatTensor([2], [4], [6])
```

Hours	Points
1	2
2	4
3	6
4	??

Hypothesis

Weight Bias

$$y = Wx + b$$

Hours	Points
1	2
2	4
3	6
4	??

Hypothesis (Cont'd)

```
x_train = torch.FloatTensor([1], [2], [3])
y_train = torch.FloatTensor([2], [4], [6])
```

```
W = torch.zeros(1, requires_grad = True)
b = torch.zeros(1, requires_grad = True)
hypothesis = x_train * W + b
```

$$y = Wx + b$$

Compute loss

- Mean Squared Error (MSE)

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Compute loss (Cont'd)

```
x_train = torch.FloatTensor([1], [2], [3])
y_train = torch.FloatTensor([2], [4], [6])
```

```
W = torch.zeros(1, requires_grad = True)
b = torch.zeros(1, requires_grad = True)
hypothesis = x_train * W + b
```

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

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Gradient descent

```
x_train = torch.FloatTensor([1], [2], [3])
y_train = torch.FloatTensor([2], [4], [6])
```

```
W = torch.zeros(1, requires_grad = True)
b = torch.zeros(1, requires_grad = True)
hypothesis = x_train * W + b
```

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

```
optimizer = optim.SGD([W, b], lr = 0.01)

optimizer.zero_grad()
cost.backward()
optimizer.step()
```

Gradient descent

```
import torch
import torch.optim as optim

# 데이터 정의
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])
# Hypothesis 초기화
W = torch.zeros(1, requires_grad=True)
b = torch.zeros(1, requires_grad=True)
# Optimizer 정의
optimizer = torch.optim.SGD([W, b], lr=0.01)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):
    # H(x) 계산
    hypothesis = x_train * W + b
    # cost 계산
    cost = torch.mean((hypothesis - y_train) ** 2)
    # Optimizer로 학습
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 100번마다 로그 출력
    if epoch % 100 == 0:
        print('Epoch {:4d}/{} W: {:.3f}, b: {:.3f} Cost: {:.6f}'.format(
            epoch, nb_epochs, W.item(), b.item(), cost.item()
        ))
```