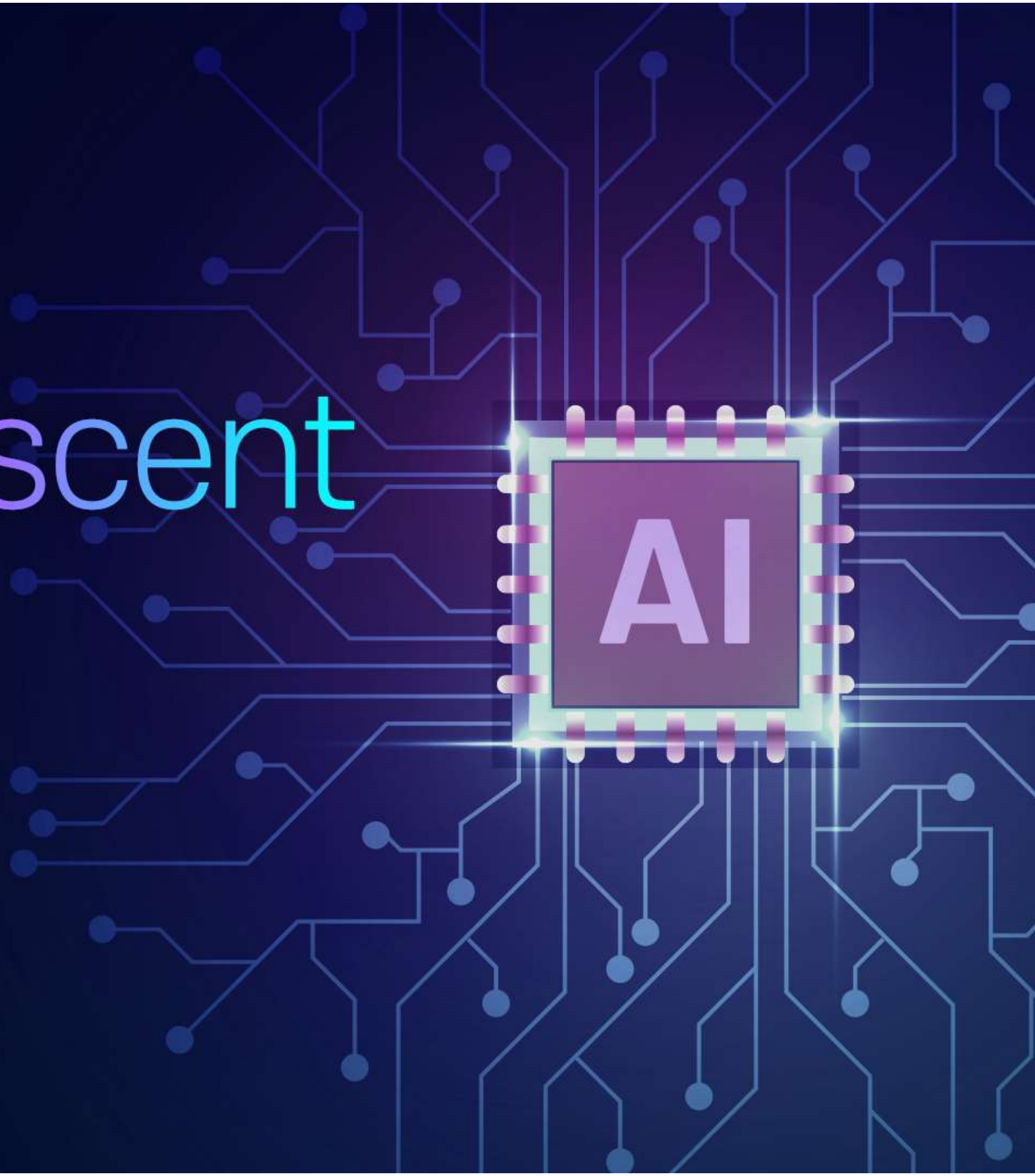
Gradient Descent (with Pytorch)

김재광 교수 (소프트웨어융합대학 글로벌융합학부)



Contents

Hypothesis Function

- Hypothesis

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

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

Hypothesis Function (Cont'd)

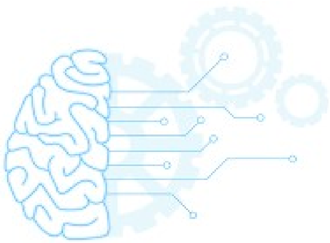
- Simper Hypothesis Function

$$H(x) = W$$

```
W = torch.zeros(1, requires_grad=True)
#b = torch.zeros(1, requires_grad=Ture)
Hypothesis = x_train * W
```

Hypothesis Function (Cont'd)

$$Input = Output!$$

1 →  → 1

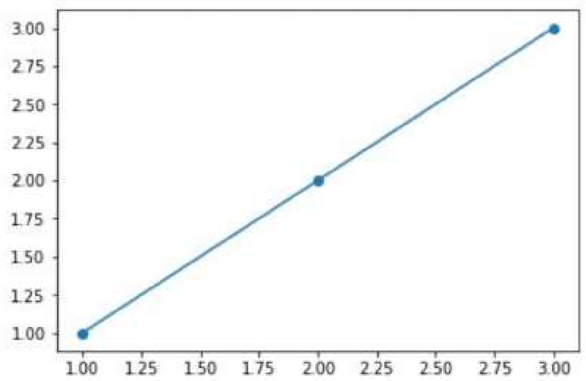
```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[1], [2], [3]])
```

Hours	Points
1	1
2	2
3	3

Hypothesis Function (Cont'd)

- **What is the best model?**
  - $H(x)=x$가 가장 정확한 모델
  - $W=1$이 가장 좋은 숫자

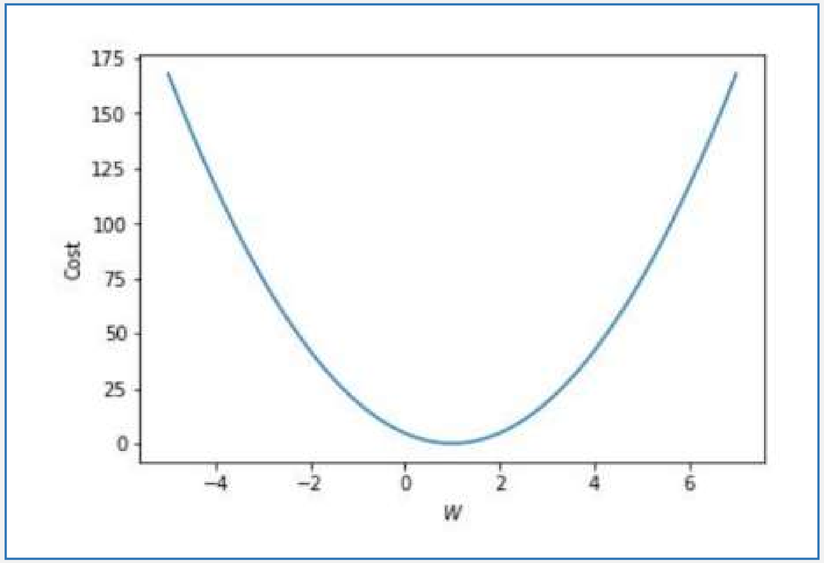
  - 어떻게 모델의 좋고 나쁨을 평가할까?



Hours	Points
1	1
2	2
3	3

Cost Function

- W=1 일 때 $cost$=0
- 1에서 멀어질수록 높아짐



Cost Function (Cont'd)

- Mean Squared Error (MSE)

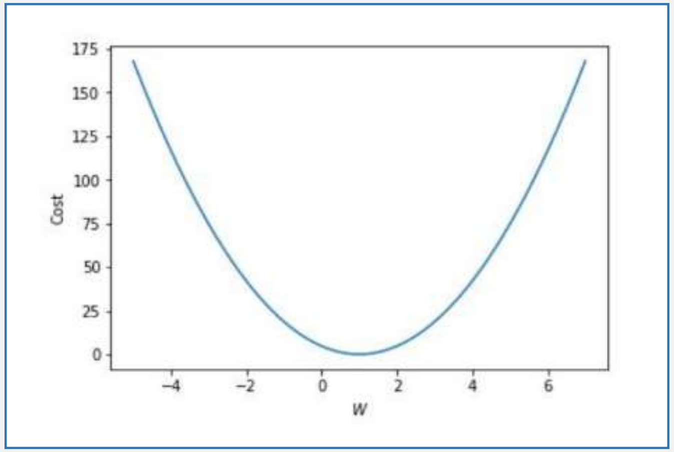
$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^{(i)}\right) - y^{(i)}\right)^2$$

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

Gradient Descent

- 최소값을 찾기 위해 곡선을 따라 내려가기
- 기울기가 크면 더 많이 움직이기
- Gradient를 계산하기

$$\frac{\partial cost}{\partial W} = \nabla W$$



Gradient Descent (Cont'd)

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

$$\nabla W \frac{\partial cost}{\partial W} = \frac{2}{m}\sum_{i=1}^{m}\left(Wx^{(i)} - y^{(i)}\right)x^{(i)}$$

$$W := W - \alpha \nabla W$$

Learning rate Gradient

Gradient Descent (Cont'd)

$$\nabla W \frac{\partial cost}{\partial W} = \frac{2}{m}\sum_{i=1}^{m}\left(Wx^{(i)} - y^{(i)}\right)x^{(i)}$$

$$W := W - \alpha \nabla W$$

```
gradient = 2*torch.mean((W*x_train – y_train)*x_train)
lr = 0.1
W -= lr*gradient
```

Implementation

```
# 데이터
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[1], [2], [3]])

# 모델 초기화
W = torch.zeros(1)
lr = 0.1

nb_epochs = 10
for epoch in range(nb_epochs + 1):
    # H(x) 계산
    hypothesis = x_train*W

    # cost and gradient 계산
    cost = torch.mean((hypothesis - y_train) ** 2)
    gradient = torch.sum((W*x_train - y_train)*x_train)

    print('Epoch {:4d}/{} W: {:.3f} Cost: {:.6f}'.format(
            epoch, nb_epochs, W.item(), cost.item()))

    W -= lr*gradient
```

Implementation (Cont'd)

- torch.optim 으로도 gradient descent를 할 수 있음
  - 시작할 때, Optimizer를 정의
  - optimizer.zero_grad() 로 gradient를 0으로 초기화
  - cost.backward() 로 gradient 계산
  - optimizer.step() 으로 gradient descent

```
# optimizer 설정
optimizer = optim.SGD([W], lr=0.1)

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

Implementation (Cont'd)

```
# 데이터
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[1], [2], [3]])

# 모델 초기화
W = torch.zeros(1, requires_grad=True)
optimizer = optim.SGD([W], lr=0.1)

nb_epochs = 10
for epoch in range(nb_epochs + 1):
    # H(x) 계산
    hypothesis = x_train*W

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

    print('Epoch {:4d}/{} W: {:.3f} Cost: {:.6f}'.format(
            epoch, nb_epochs, W.item(), cost.item()))
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()
```