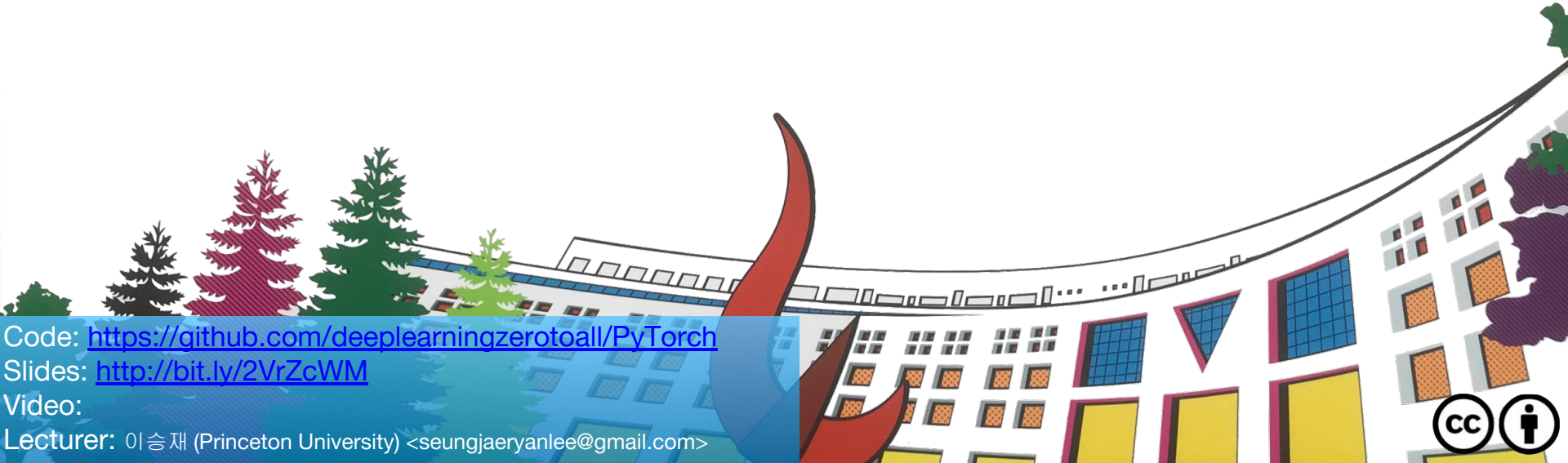
ML/DL for Everyone Season 2

with PYTORCH

Deeper Look at Gradient Descent



Code: https://github.com/deeplearningzerotoall/PyTorch
Slides: http://bit.ly/2VrZcWM
Video:
Lecturer: 이승재 (Princeton University) <seungjaeryanlee@gmail.com>

Deeper Look at Gradient Descent

- Hypothesis function 복습
- 사용할 모의 data 확인
- Cost function 이해
- Gradient descent 이론
- Gradient descent 구현
- Gradient descent 구현 (nn.optim)

Hypothesis (Linear Regression)

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

Weight

Bias

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

Simpler Hypothesis Function

$$H(x) = Wx$$

Weight

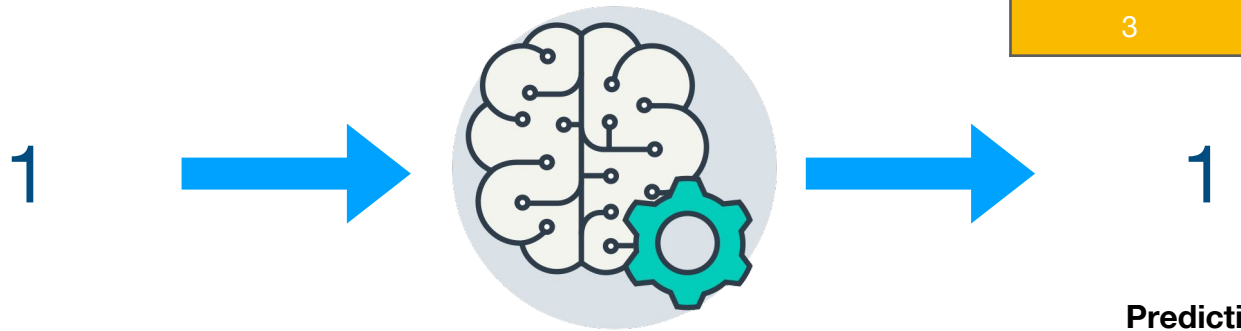
No Bias!

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

Dummy Data

Input = Output!

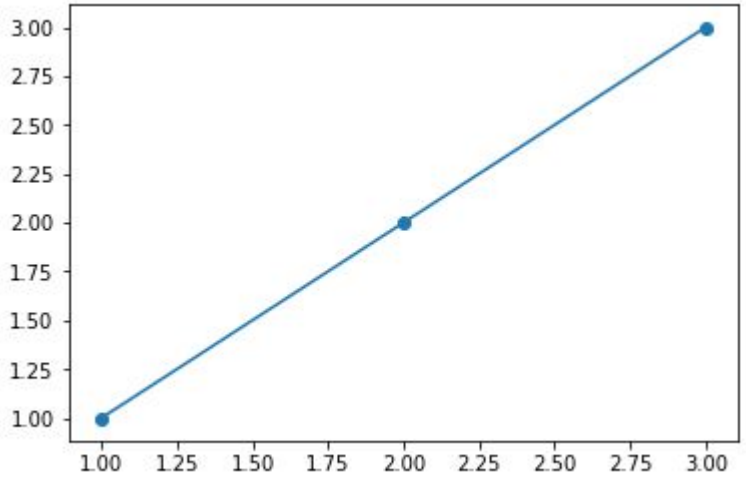
Hours (x)	Points (y)
1	1
2	2
3	3




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

What is the best model?

- $H(x) = x$ 가 정확한 모델
- $W = 1$이 가장 좋은 숫자



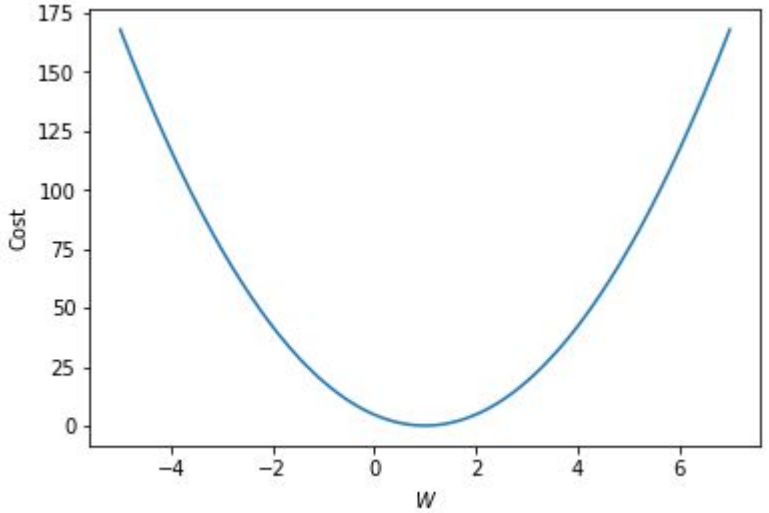
어떻게 **모델의 좋고 나쁨**을

평가할 수 있을까요?

Hours (x)	Points (y)
1	1
2	2
3	3

Cost function: Intuition

- $W = 1$ 일 때 $cost = 0$
- 1 에서 멀어질수록 높아진다.



Cost function: MSE

Mean Squared Error (MSE)

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

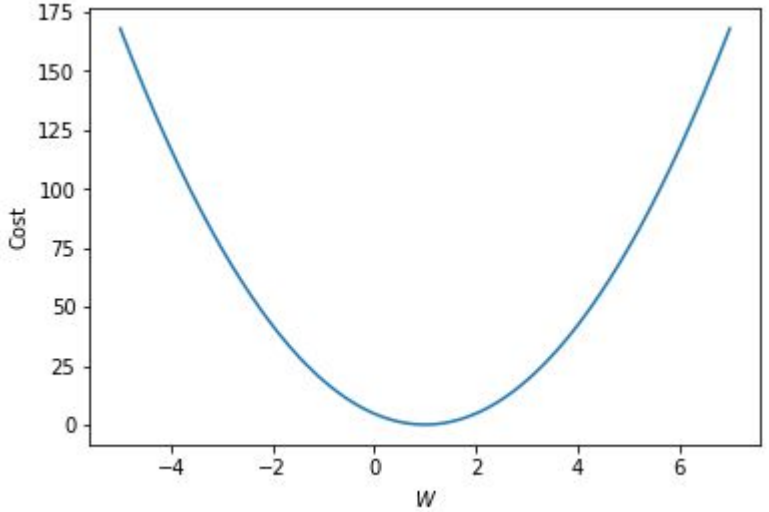
Mean Prediction Target

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

Gradient Descent: Intuition

- 곡선을 내려가자
- 기울기가 클수록 더 멀리!
- **“Gradient”** 를 계산하자

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



Gradient Descent: The Math

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

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

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

Learning rate

Gradient

Gradient Descent: Code

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

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

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

Full Code

```
# 데이터
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[1], [2], [3]])
# 모델 초기화
W = torch.zeros(1)
# learning rate 설정
lr = 0.1

nb_epochs = 10
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = x_train * W

    # cost 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()
    ))

    # cost gradient로 H(x) 개선
    W -= lr * gradient
```

결과

```
Epoch    0/10 W: 0.000, Cost: 4.666667
Epoch    1/10 W: 1.400, Cost: 0.746666
Epoch    2/10 W: 0.840, Cost: 0.119467
Epoch    3/10 W: 1.064, Cost: 0.019115
Epoch    4/10 W: 0.974, Cost: 0.003058
Epoch    5/10 W: 1.010, Cost: 0.000489
Epoch    6/10 W: 0.996, Cost: 0.000078
Epoch    7/10 W: 1.002, Cost: 0.000013
Epoch    8/10 W: 0.999, Cost: 0.000002
Epoch    9/10 W: 1.000, Cost: 0.000000
Epoch   10/10 W: 1.000, Cost: 0.000000
```

- **Epoch**: 데이터로 학습한 횟수
- 학습하면서 점점:
  - 1에 수렴하는 W
  - 줄어드는 cost

Gradient Descent with `torch.optim`

- torch.optim 으로도 gradient descent 를 할 수 있답니다!
  - 시작할 때 `Optimizer` 정의
  - `optimizer.zero_grad()` 로 gradient 를 0으로 초기화
  - `cost.backward()` 로 gradient 계산
  - `optimizer.step()` 으로 gradient descent

```
# optimizer 설정
optimizer = optim.SGD([W], lr=0.15)

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

Full Code with `torch.optim`

```
# 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 설정
optimizer = optim.SGD([W], lr=0.15)

nb_epochs = 10
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = x_train * W

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

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

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

결과

```
Epoch    0/10 W: 0.000, Cost: 4.666667
Epoch    1/10 W: 1.400, Cost: 0.746666
Epoch    2/10 W: 0.840, Cost: 0.119467
Epoch    3/10 W: 1.064, Cost: 0.019115
Epoch    4/10 W: 0.974, Cost: 0.003058
Epoch    5/10 W: 1.010, Cost: 0.000489
Epoch    6/10 W: 0.996, Cost: 0.000078
Epoch    7/10 W: 1.002, Cost: 0.000013
Epoch    8/10 W: 0.999, Cost: 0.000002
Epoch    9/10 W: 1.000, Cost: 0.000000
Epoch   10/10 W: 1.000, Cost: 0.000000
```

- 학습하면서 점점:
  - 1에 수렴하는 W
  - 줄어드는 cost

What’s Next?

- 지금까지 하나의 정보로부터 추측하는 모델을 만들었습니다.
  - 예시 1) 수업 참여도 -> 수업 점수
  - 예시 2) 총 수면 시간 -> 집중력
- 하지만 대부분의 추측은 많은 정보를 추합해서 이뤄집니다.
  - 예시 1) 쪽지 시험 성적들 -> 중간고사 성적
  - 예시 2) 암의 위치, 넓이, 모양 -> 치료 성공률
- 여러 개의 정보로부터 결론을 추측하는 모델은 어떻게 만들까요?