Automatic Writing Evaluation on Regression

MLP LAB 임경태, 임현석

These slides created with reference to ML for Everyone Season2 with Pytorch

Contents

01 Basic concept of ML

02 Linear Regression on Pytorch

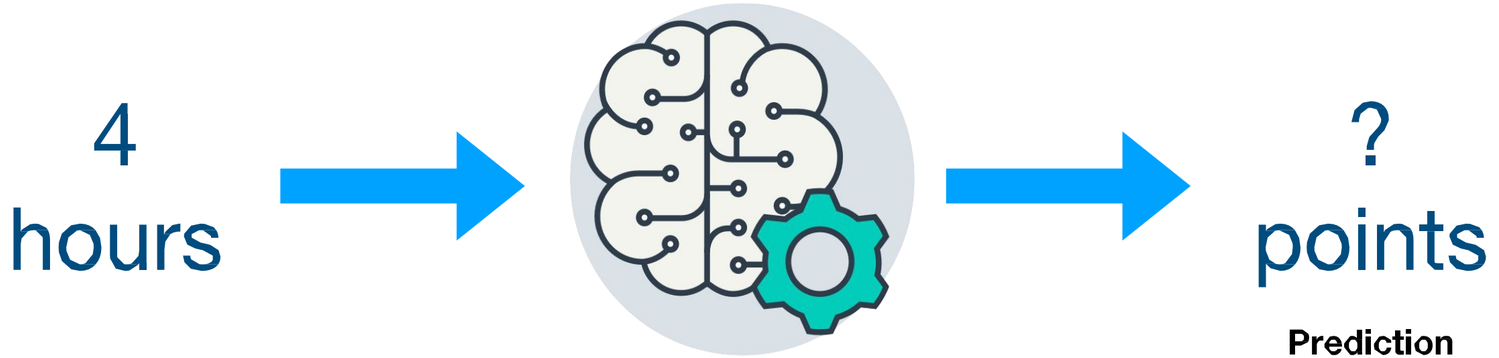
03 Multi-variable LR

04 Batch based Gradient Descent

01. 가장 간단한 기계학습

Problem definition

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



Hours (x)	Points (y)
1	2
2	4
3	6
4	?

Training dataset

Test dataset

Preparing Dataset

- Let's just convert the data to a Pytorch format

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

$$X_{\text{train}} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad Y_{\text{train}} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

Hours (x)	Points (y)
1	2
2	4
3	6

- 데이터는 `torch.tensor` !
- 입력 따로 , 출력 따로 !
  - 입력 : `x_train`
  - 출력 : `y_train`
  - 입출력은 x, y 로 구분

A question

- 그래서 컴퓨터에게 **현실 문제**를 어떻게 알려 줄거야?

그래 뭐 딱 보아하니 함수 만들면 되겠네 ㅋ
공부시간 * 가중치 = 점수

A solution: 가설함수

그래 뭐 딱 보아하니 함수 만들면 되겠네 ㅋ
공부시간 * 가중치 = 점수

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

```
def f(x):
    y = w*x
    return y
```

```
for a_sample in x_train:
    prediction = f(a_sample)
    print(prediction)
```

A solution: Loss함수

내가 만든 함수는 예측을 얼마나 잘 하고 있나?
뇌피셜로 할순 없지!

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

```
def f(x):
    y = w*x
    return y
```

```
def loss(predicted, target):
    return predicted - target
```

```
loss_value = 0
for a_sample in x_train:
    prediction = f(a_sample)
    loss_value += loss(prediction, y_train)
```

문제점

어라 그런데 학습 데이터가 추가되었네?

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

```
def f(x):
    y = w*x
    return y
```

```
def loss(predicted, target):
    return predicted - target
```

```
loss_value = 0
for a_sample in x_train:
    prediction = f(a_sample)
    loss_value += loss(prediction, y_train)
```

Hours (x)	Points (y)
1	2
2	4
3	6
1	3
3	5
8	7

근본적인 해결방법: 기계학습의 개념

W값을 데이터에 맞게 자동으로 선택해주면 안되나?

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

```
def f(x):
    y = w*x
    return y
```

```
def loss(predicted, target):
    return predicted - target
```

```
loss_value = 0
for a_sample in x_train:
    prediction = f(a_sample)
    loss_value += loss(prediction, y_train)
```

Hours (x)	Points (y)
1	2
2	4
3	6
1	3
3	5
8	7

근본적인 해결방법: 기계학습-지도학습의 도입

관측 데이터에 따라 최적의 W값을 찾는 여정을 떠나보자
일단 우리가 짠 함수를 살펴볼까?

$$y = Wx + b$$

Weight Bias

```
def f(x):
    y = w*x + b
    return y
```

```
def loss(predicted, target):
    return (predicted - target)**2
```

Hours (x)	Points (y)
1	2
2	4
3	6
1	3
3	5
8	7

기계학습-지도학습의 도입 with 무작위 선택

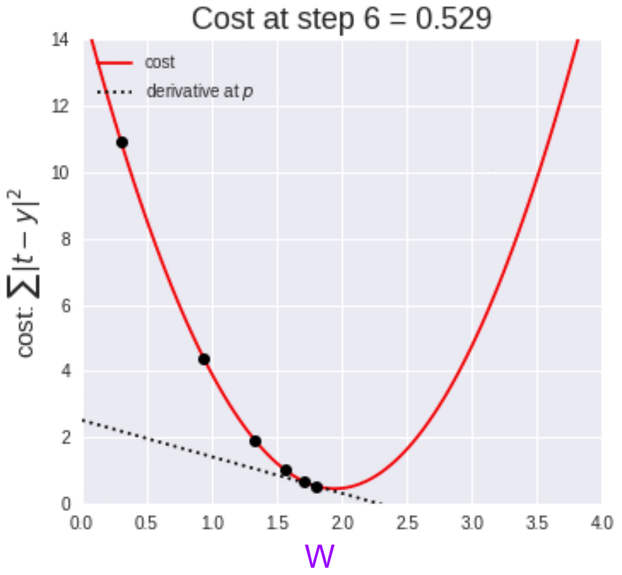
임의에 W를 넣었을 때 loss값이 0이 되는 W값이 최적이겠네?
그렇다면! W값 대비 loss값을 그래프로 그려보자 W==2일 때?

$$y = Wx + b$$

Weight Bias

```
def f(x):
    y = w*x + b
    return y
```

```
def loss(predicted, target):
    return predicted - target
```



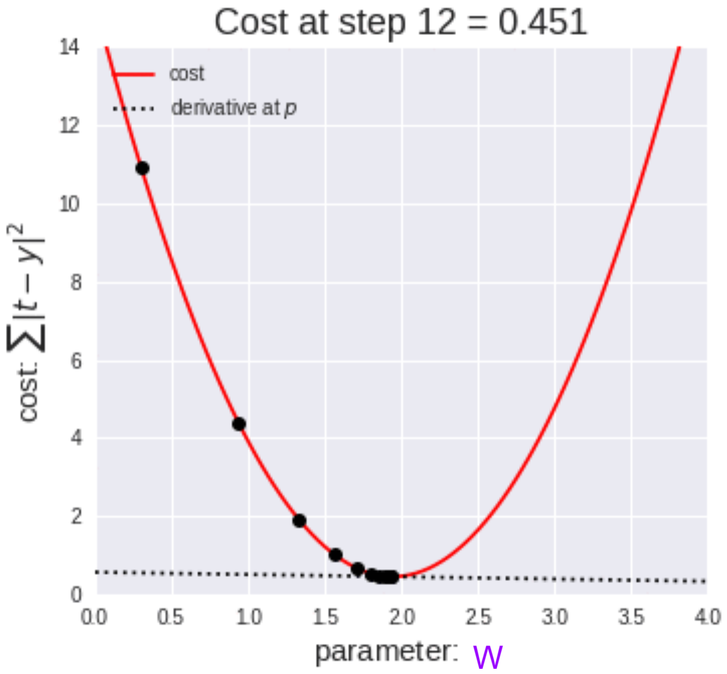
Hours (x)	Points (y)
1	2
2	4
3	6

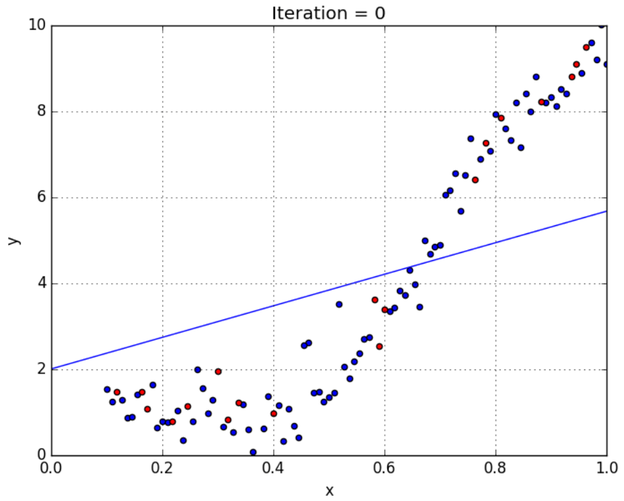
Gradient Descent

그런데 무한정 W값을 랜덤으로 다 넣어서 돌려볼 수 있을까?

- W값이 loss_value에 미치는 영향을 토대로 W값을 업데이트 하자!

  - 기울기가 클수록 더 멀리! 곡선을 내려가자! 그러기위해 “Gradient” 를 계산하자 $\frac{\partial cost}{\partial W} = \nabla W$





REF: https://medium.com/analytics-vidhya/simple-linear-regression-cost-function-gradient-descent-50c5ed085770

Gradient Descent

그런데 무한정 W값을 랜덤으로 다 넣어서 돌려볼 수 있을까?

- 기울기가 클수록 더 멀리! 곡선을 내려가자! 그러기위해 “Gradient” 를 계산하자 $\frac{\partial cost}{\partial W} = \nabla W$

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

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

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

Learning rate

Gradient

Gradient Descent

- 코드로 짜보자

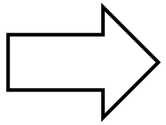
  - 기울기가 클수록 더 멀리! 곡선을 내려가자! 그러기 위해 “Gradient”를 계산하자

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

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

```
def f(x):
    y = w*x
    return y
```

```
def loss(predicted, target):
    return (predicted - target)**2
```



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

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

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

Training

- 코드로 짜보자

```
# 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
```

```
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:
  - 정답 1에 수렴하는 W
  - 줄어드는 cost

02. Pytorch를 활용한 기계학습

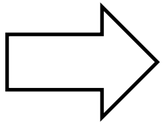
Gradient Descent구현의 문제점

- 가설함수를 2차원함수 $f(x) = Wx^2$으로 구현할 경우.. Gradient 계산 코드를 바꿔야해!
- 가설함수 f(x)를 계속 변경하더라도 자동으로 미분값을 구해주면 좋을 텐데…

Gradient Descent with torch.optim

- torch.optim으로 미분값을 자동으로 구하고 gradient descent를 할 수 있음
  - 시작할 때 Optimizer 정의
  - optimizer.zero_grad() 로 gradient 를 0으로 초기화
  - cost.backward() 로 gradient 계산
  - optimizer.step() 으로 gradient descent

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



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

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

Gradient Descent 구현 비교

```
# 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
```

```
# 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()
```

인생은 **실전**

- 자동 글쓰기평가 Automatic Writing Evaluation (AWE) 시스템을 만들자!

- 1) AWE_1000_score.csv 파일을 읽고

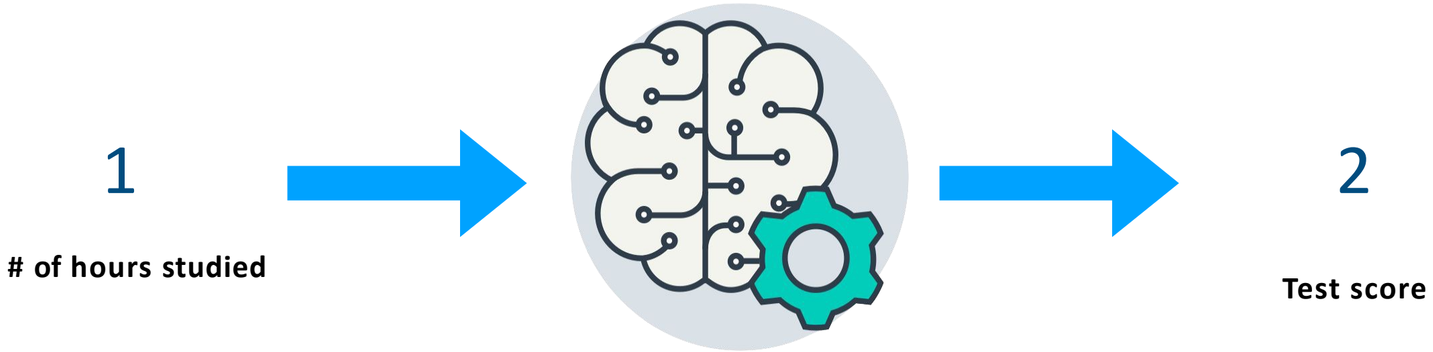
- 2) content 컬럼의 글쓰기에 활용된 문장 수를 계산해 sentence 컬럼에 문장의 수를 저장하자!

- 3) 문장의 수를 입력으로 해당 글의 글쓰기 평가점수를 예측하는 모델을 구현 하시오.

	id	nationality	sex	title	content	phrase	sentence
0	A100000_v01.txt_1.txt	중국	남자	사진기 빌리기	하지만 빌리씨하고 나오코씨는 모두 사진기가 없었어요. 그래서 빌리씨는 모하메드 씨...	1.0	5.0

03. Multivariable Linear Model

Single Variable Linear Regression

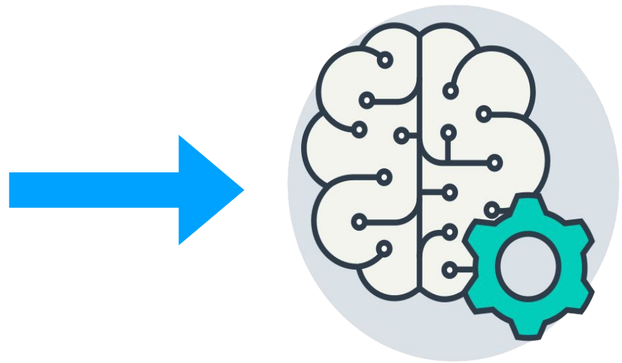


$$H(x) = Wx + b$$

Multi-Variable Linear Regression

Quiz1: 73
Quiz2: 80
Quiz3: 75

Quiz Scores



152

Final Score Prediction

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

Multi-Variable Linear Regression

- Dataset

Quiz 1 (x1)	Quiz 2 (x2)	Quiz 3 (x3)	Final (y)
73	80	75	152
93	88	93	185
89	91	80	180
96	98	100	196
73	66	70	142

```
                          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]])
```

Multi-Variable Linear Regression

- Hypothesis Function

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

x 라는 vector 와 W 라는 matrix의 곱

$$H(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

입력변수가 3개라면 weight 도 3개!

Multi-Variable Linear Regression

- Hypothesis Function

```
# H(x) 계산
hypothesis = x1_train * w1 + x2_train * w2 + x3_train * w3 + b
```

- 단순한 hypothesis 정의!
- 하지만 x 가 길이 1000의 vector라면...?

$$H(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Multi-Variable Linear Regression

- Hypothesis Function

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

- matmul()로 한번에 계산
  a. 더 간결하고,
  b. x의 길이가 바뀌어도 코드를 바꿀 필요가 없고
  c. 속도도 더 빠르다!

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

Multi-Variable Linear Regression

- Cost Function: 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)
```

Multi-Variable Linear Regression

- Cost Function: MSE

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

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

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

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

Multi-Variable Linear Regression

- Full code with Pytorch

```
# 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]])
```

1. 데이터 정의

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

2. 모델 정의

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

3. optimizer 정의

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

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

4. Hypothesis 계산

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

5. Cost 계산 (MSE)

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

6. Gradient descent

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

Multi-Variable Linear Regression

- Full code with Pytorch

```
Epoch    0/20 hypothesis: tensor([0., 0., 0., 0., 0.]) Cost: 29661.800781
Epoch    1/20 hypothesis: tensor([67.2578, 80.8397, 79.6523, 86.7394, 61.6605]) Cost: 9298.520508
Epoch    2/20 hypothesis: tensor([104.9128, 126.0990, 124.2466, 135.3015,  96.1821]) Cost: 2915.713135
Epoch    3/20 hypothesis: tensor([125.9942, 151.4381, 149.2133, 162.4896, 115.5097]) Cost: 915.040527
Epoch    4/20 hypothesis: tensor([137.7968, 165.6247, 163.1911, 177.7112, 126.3307]) Cost: 287.936005
Epoch    5/20 hypothesis: tensor([144.4044, 173.5674, 171.0168, 186.2332, 132.3891]) Cost: 91.371017
Epoch    6/20 hypothesis: tensor([148.1035, 178.0144, 175.3980, 191.0042, 135.7812]) Cost: 29.758139
Epoch    7/20 hypothesis: tensor([150.1744, 180.5042, 177.8508, 193.6753, 137.6805]) Cost: 10.445305
Epoch    8/20 hypothesis: tensor([151.3336, 181.8983, 179.2240, 195.1707, 138.7440]) Cost: 4.391228
Epoch    9/20 hypothesis: tensor([151.9824, 182.6789, 179.9928, 196.0079, 139.3396]) Cost: 2.493135
Epoch   10/20 hypothesis: tensor([152.3454, 183.1161, 180.4231, 196.4765, 139.6732]) Cost: 1.897688
Epoch   11/20 hypothesis: tensor([152.5485, 183.3610, 180.6640, 196.7389, 139.8602]) Cost: 1.710541
Epoch   12/20 hypothesis: tensor([152.6620, 183.4982, 180.7988, 196.8857, 139.9651]) Cost: 1.651413
Epoch   13/20 hypothesis: tensor([152.7253, 183.5752, 180.8742, 196.9678, 140.0240]) Cost: 1.632387
Epoch   14/20 hypothesis: tensor([152.7606, 183.6184, 180.9164, 197.0138, 140.0571]) Cost: 1.625923
Epoch   15/20 hypothesis: tensor([152.7802, 183.6427, 180.9399, 197.0395, 140.0759]) Cost: 1.623412
Epoch   16/20 hypothesis: tensor([152.7909, 183.6565, 180.9530, 197.0538, 140.0865]) Cost: 1.622141
Epoch   17/20 hypothesis: tensor([152.7968, 183.6643, 180.9603, 197.0618, 140.0927]) Cost: 1.621253
Epoch   18/20 hypothesis: tensor([152.7999, 183.6688, 180.9644, 197.0662, 140.0963]) Cost: 1.620500
Epoch   19/20 hypothesis: tensor([152.8014, 183.6715, 180.9666, 197.0686, 140.0985]) Cost: 1.619770
Epoch   20/20 hypothesis: tensor([152.8020, 183.6731, 180.9677, 197.0699, 140.1000]) Cost: 1.619033
```

Final (y)
152
185
180
196
142

- 점점 작아지는 Cost
- 점점 y 에 가까워지는 H(x)
- Learning rate 에 따라 발산할수도!

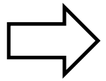
Multi-Variable Linear Regression

- 코드 효율화: nn.Module 활용

  - nn.Mudule 을 상속받은 class내 모든 parameters들은 자동으로 미분을 계산한다!

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

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



```
import torch.nn as nn

class MultivariateLinearRegressionModel(nn.Module):
  def _init_(self):
    super()._init_()
    self.linear = nn.Linear(3, 1)

  def forward(self, x):
    return self.linear(x)

hypothesis = model(x_train)
```

- nn.Module 을 상속해서 모델 생성
- nn.Linear(3, 1)
  - 입력 차원: 3
  - 출력 차원: 1
- Hypothesis 계산은 forward() 에서!
- Gradient 계산은 PyTorch 가 알아서 해준다 backward()

Multi-Variable Linear Regression

- 코드 효율화: F.mse_loss 활용

 - Loss함수가 이미 구현되어 있다고?

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

```
import torch.nn.functional as F

# cost 계산
cost = F.mse_loss(prediction, y_train)
```

- torch.nn.functional 에서 제공하는 loss function 사용
- 쉽게 다른 loss와 교체 가능! (l1_loss, smooth_l1_loss 등…)

Multi-Variable Linear Regression

- 간결한 코드 정리

```
# 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)
model = MultivariateLinearRegressionModel()


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

```
import torch.nn as nn

class MultivariateLinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(3, 1)

    def forward(self, x):
        return self.linear(x)
```

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

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

  # cost 계산
  cost = torch.mean((hypothesis - y_train) ** 2)
  cost = F.mse_loss(prediction, y_train)

  # 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()
  ))
```

인생은 **실전**

- 자동 글쓰기평가 Automatic Writing Evaluation (AWE) 시스템을 만들자!

~~- 1) AWE_1000_score.csv 파일을 읽고~~

~~- 2) content 컬럼의 글쓰기에 활용된 문장 수를 계산해 sentence 컬럼에 문장의 수를 저장하자!~~

~~- 3) 문장의 수를 입력으로 해당 글의 글쓰기 평가점수를 예측하는 모델을 구현 하시오.~~

- 4) 문장의 수와 phrase 수를 입력으로 해당 글의 글쓰기 평가점수를 예측하는 모델을 구현 하시오. (phrase 수는 어떻게 셀까?)

	id	nationality	sex	title	content	phrase	sentence
0	A100000_v01.txt_1.txt	중국	남자	사진기 빌리기	하지만 빌리씨하고 나오코씨는 모두 사진기가 없었어요. 그래서 빌리씨는 모하메드 씨...	1.0	5.0

04. Data batch 처리

Big Data 문제의 발생

- 자동 글쓰기평가 Automatic Writing Evaluation (AWE) 시스템을 만들 때!
- 1000개의 학습데이터를 한번에 연산해서 Loss를 구한 후 W를 업데이트 한다??
  - 학습데이터가 1,000,000개가 된다면?? 한번 업데이트 할 때 한세월?
  - 병렬 연산으로 진행해야 하는데 이것도 무리데스

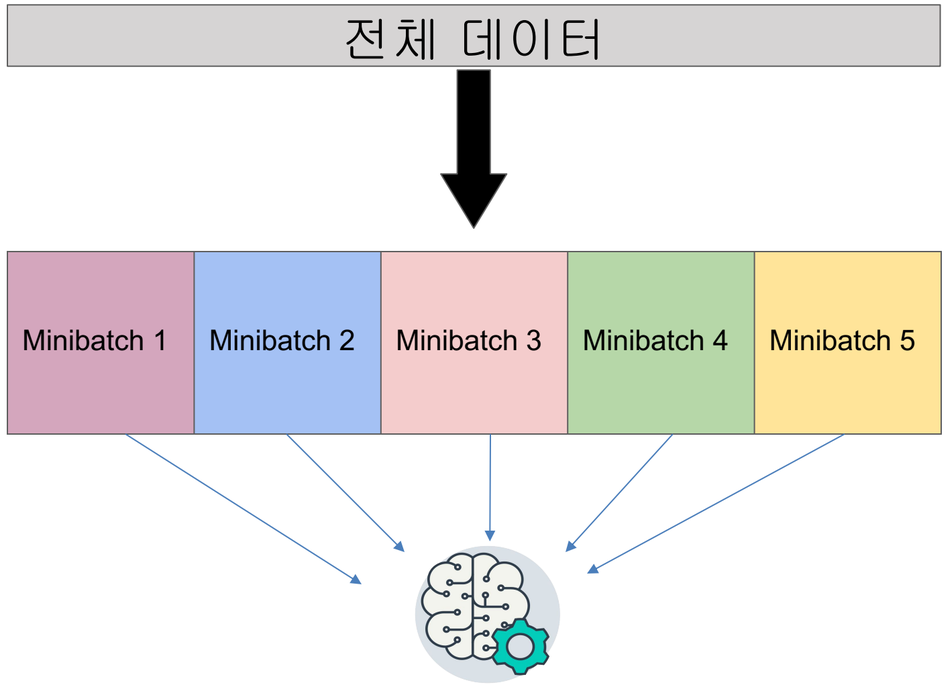
$$y = Wx + b$$

Weight Bias

X의 차원이 [1000000x2] 라고하면 이걸 한번에 연산할 수 있을까?

Big Data 문제의 해결

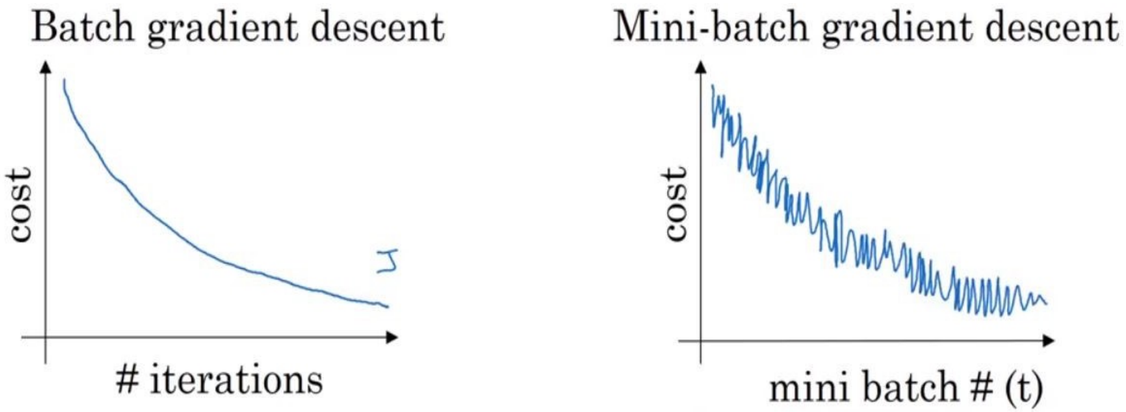
- Minibatch Gradient Descent: 데이터를 나눠서 모델을 학습하자



Big Data 문제의 해결

- Minibatch Gradient Descent: 데이터를 나눠서 모델을 학습하자

  - 업데이트를 좀 더 빠르게 할 수 있다.
  - 전체 데이터를 쓰지 않아서 잘못된 방향으로 업데이트를 할 수도 있다.



https://engmrk.com/mini-batch-gd/

Minibatch on Pytorch

- Minibatch Gradient Descent: 데이터를 나눠서 모델을 학습하자

```
                from torch.utils.data import Dataset

class CustomDataset(Dataset): def __init__(self):
                        self.x_data = [[73, 80, 75],
                                       [93, 88, 93],
                                       [89, 91, 90],
                                      [96, 98, 100],
                                       [73, 66, 70]]
    self.y_data = [[152], [185], [180], [196], [142]]

  def __len__(self):
    return len(self.x_data)

  def __getitem__(self, idx):
    x = torch.FloatTensor(self.x_data[idx])
    y = torch.FloatTensor(self.y_data[idx])

    return x, y


dataset = CustomDataset()
```

- `torch.utils.data.Dataset` 상속
- `__len__()`
  - 이 데이터셋의 총 데이터 수
- `__getitem__()`
  - 어떠한 인덱스 **idx** 를 받았을 때, 그에 상응하는 입출력 데이터 반환

Minibatch on Pytorch

- Minibatch Gradient Descent: 데이터를 나눠서 모델을 학습하자

```
from torch.utils.data import DataLoader



dataloader = DataLoader( dataset,
  batch_size=2, shuffle=True,
)
```

- `torch.utils.data.DataLoader` 사용
- `batch_size=2`
  - 각 minibatch의 크기
  - 통상적으로 2의 제곱수로 설정한다
    (16, 32, 64, 128, 256, 512…)
- `shuffle=True`
  - Epoch 마다 데이터셋을 섞어서, 데이터가 학습되는 순서를 바꾼다.

Minibatch on Pytorch

- Minibatch Gradient Descent: 데이터를 나눠서 모델을 학습하자

```
nb_epochs = 20
for epoch in range(nb_epochs + 1):
  for batch_idx, samples in enumerate(dataloader):
    x_train, y_train = samples
    # H(x) 계산
    prediction = model(x_train)

    # cost 계산
    cost = F.mse_loss(prediction, y_train)

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

    print('Epoch {:4d}/{} Batch {}/{} Cost: {:.6f}'.format(
      epoch, nb_epochs, batch_idx+1, len(dataloader),
      cost.item()
    ))
```

- `enumerate(dataloader)`
  - minibatch 인덱스와 데이터를 받음.

- `len(dataloader)`
  - 한 epoch당 minibatch 개수

실습

- 자동 글쓰기평가 Automatic Writing Evaluation (AWE) 시스템을 만들자!
  - 1) AWE_1000_score.csv 파일을 읽고
  - 2) content 컬럼의 글쓰기를 batch_size 4개씩 가져오는 Dataloader를 구현하시오
  - 3) 위에 정의한 Dataloader의 출력값을 문장의 one-hot encoding으로 반환받게 만드시오

인생은 **실전**

- 자동 글쓰기평가 Automatic Writing Evaluation (AWE) 시스템을 만들자!

  - 1) AWE_1000_score.csv 파일을 읽고

  - 2) content 컬럼의 글쓰기를 입력으로 점수를 예측하는 모델을 구현 하시오

- 심화: 한국어 글쓰기 평가를 위해 추가로 어떤 feature들을 활용할 수 있지 않을까?

  - 가령 어려운 단어를 많이 썼다든지..?

감사합니다.