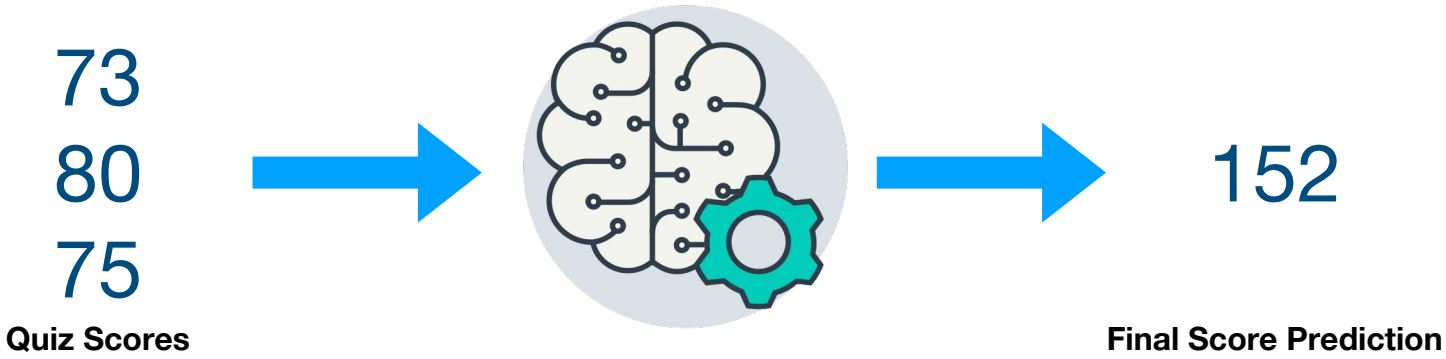
Loading Data 목차

- Multivariate Linear Regression 복습
- “Minibatch” Gradient Descent 이론
- PyTorch Dataset and DataLoader 사용법

Multivariate Linear Regression



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

Data

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]])
```

Results

```
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!

Data in the Real World

- 복잡한 머신러닝 모델을 학습하려면 엄청난 양의 데이터가 필요하다!
- 대부분 데이터셋은 적어도 수십만 개의 데이터를 제공한다.

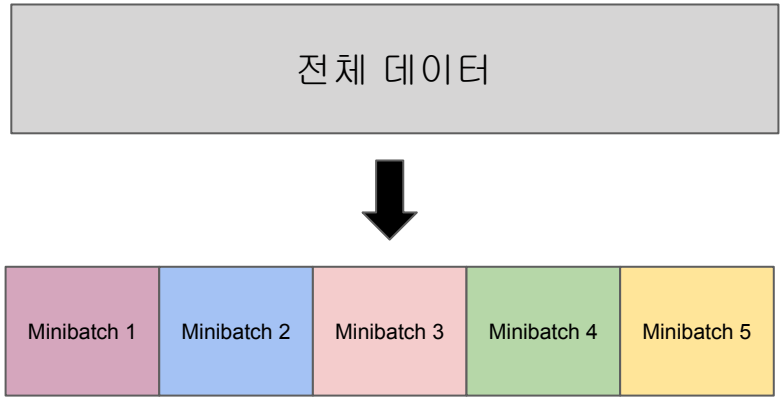
자그마치 14,197,122 개의 이미지

Data in the Real World: Problem

- 엄청난 양의 데이터를 한번에 학습시킬 수 없다!
  - 너무 느리다.
  - 하드웨어적으로 불가능하다.
- 그렇다면 일부분의 데이터로만 학습하면 어떨까?

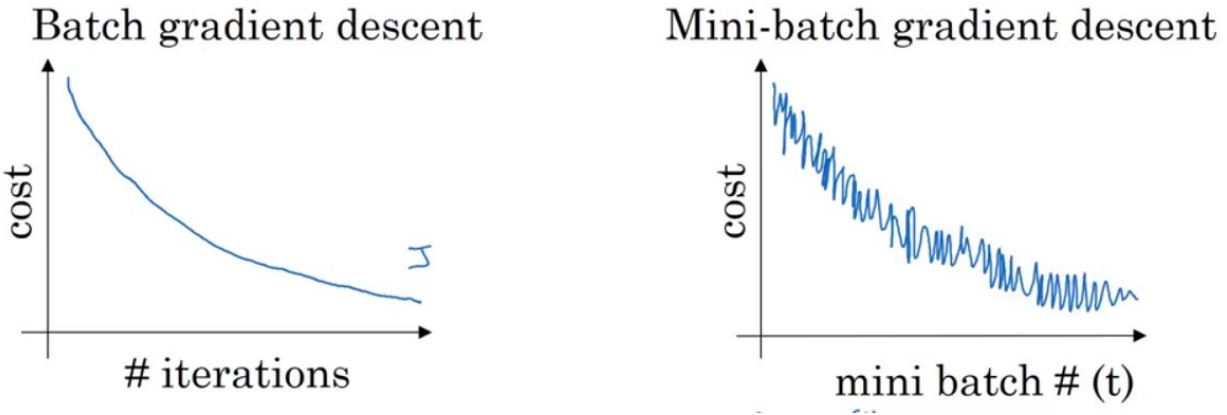
Minibatch Gradient Descent

- 전체 데이터를 균일하게 나눠서 학습하자!



Minibatch Gradient Descent: Effects

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



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

PyTorch Dataset

```
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

PyTorch DataLoader

```
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.

Full Code with Dataset and DataLoader

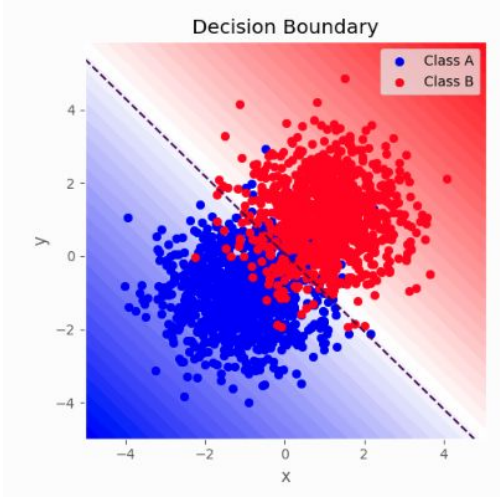
```
nb_epochs = 20
for epoch in range(nb_epochs + 1):
    for batch_idx, samples in enumerate(dataloader):
        x_train, y_train = samples
        # H(x) 계산
        prediction = model(x_train)

        # 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

What's Next?

- 지금까지는 어떠한 숫자 하나를 예측하는 모델을 만들었다.
- 분류하는 모델은 어떻게 만들 수 있을까?



http://scikit-hep.org/root_numpy/auto_examples/tmva/plot_twoclass.html