Loading Data

상태	Basic ML
담당자	

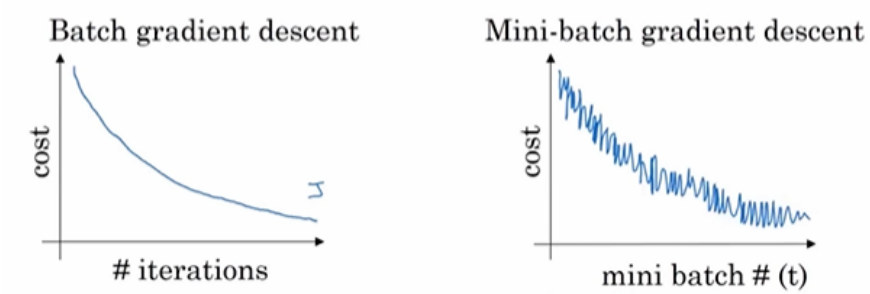
Data in the Real World

현실 세계에서 복잡한 모델을 학습하려면 엄청난 양의 데이터가 필요하다. 이 데이터를 한번에 학습시키는 것은 1. 너무 느리고 2. 하드웨어적으로 불가능하다.

데이터의 일부분으로만 학습하는 것은 어떨까? “Minibatch Gradient Descent”

Minibatch Gradient Descent

전체 데이터를 균일하게 나누어, 컴퓨터는 각 minibatch별로 gradient descent를 수행하게 된다.



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

1단계 : Dataset

```
class CustomDataset(Dataset):
    def __init__(self):
        self.x_data = [[70, 80, 75],
                       [93, 88, 93],
                       [89, 91, 99],
                       [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): # 데이터에 필요한 format을 맞춰준다 e.g. normalize, transform
        x = torch.FloatTensor(self.x_data[idx])
        y = torch.FloatTensor(self.y_data[idx])

        return x, y
```

```
dataset = CustomDataset()
```

- `__len__()`을 구현했기 때문에 `len()`을 이용할 수 있다.
- `__getitem__()`을 구현했기 때문에 인덱스에 접근할 수 있다.
- 보통 `__init_`, `__len__`, `__getitem__` 스페셜 메서드 정도를 커스텀 데이터셋에 정의한다.
- 클래스에 스페셜 메소드로 `__len__` 등을 구현해 주었기 때문에, 클래스의 인스턴스인 `dataset`에서도 `len()` 함수 등을 쓸 수 있다.
    - 평소에 List 류의 객체를 쓸 때에는 바로 `len()`을 적용해도 문제가 없었는데, 이건 `list` 클래스에 `__len__`이 구현되어 있었기 때문

youtube-cnn-007-pytorch-cyclegan/dataset.py at master · hanyoseob/youtube-cnn-007-pytorch-cyclegan

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters. Learn more about bidirectional Unicode characters You can't perform that action at this time. You signed in with another tab or window.

https://github.com/hanyoseob/youtube-cnn-007-pytorch-cyclegan/blob/master/dataset.py

hanyoseob/**youtube-cnn-007-pytorch-cyclegan**

[CNN PROGRAMMING] 007 - CycleGAN

1 Contributor 0 Issues 2 Stars 3 Forks

2단계 : DataLoader

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

- `batch_size` : minibatch의 크기로, 통상적으로 2의 제곱수로 설정
- `shuffle=True` : Epoch마다 데이터셋을 섞어 데이터가 학습되는 순서를 변경

3단계 : 학습

모든 학습 데이터를 한 번 돌 때 '한 에포크' 학습했다고 말한다.

```
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
model = MultivariateLinearRegressionModel()
optimizer = optim.SGD(model.parameters(), lr=1e-5)

for epoch in range(1, nb_epochs + 1):
    for batch_idx, samples in enumerate(dataloader): # 미니배치의 인덱스와 데이터를 받고, 이 인덱스를 x, y로 나눔
        x_train, y_train = samples # enumerate에서 인덱스를 쓰기 위해 __getitem__ 호출, 이것이 x와 y를 리턴
        # Prediction
        prediction = model(x_train)
        # Cost
        cost = F.mse_loss(prediction, y_train)
        # Update prediction
        optimizer.zero_grad()
        cost.backward()
        optimizer.step()
        print("Epoch {:4d}/{} Batch {}/{} Cost : {:.6f}".format(
```

```
        epoch, nb_epochs, batch_idx+1, len(dataloader), cost.item()))

>>

Epoch    1/20 Batch 1/3 Cost : 5430.643066
Epoch    1/20 Batch 2/3 Cost : 2690.722168
Epoch    1/20 Batch 3/3 Cost : 407.178223
Epoch    2/20 Batch 1/3 Cost : 256.779358
Epoch    2/20 Batch 2/3 Cost : 31.873177
Epoch    2/20 Batch 3/3 Cost : 14.551682
...
Epoch   19/20 Batch 1/3 Cost : 5.917408
Epoch   19/20 Batch 2/3 Cost : 2.799996
Epoch   19/20 Batch 3/3 Cost : 47.471249
Epoch   20/20 Batch 1/3 Cost : 4.965910
Epoch   20/20 Batch 2/3 Cost : 18.682472
Epoch   20/20 Batch 3/3 Cost : 20.126045
```