Homework 3
Comparison of SGD and GD

2021050300
김재민

1. Train Shallow Neural Networks using both SGD and GD

1. XOR 데이터 생성 (NumPy 사용)

```
# [1] 라이브러리 import
import numpy as np
import torch
import torch.optim as optim
import pandas as pd
import matplotlib.pyplot as plt
```

```
# [2] XOR 데이터 정의
X_seeds = np.array([[0, 0], [1, 0], [0, 1], [1, 1]], dtype=np.float32)
y_seeds = np.array([0, 1, 1, 0])  # XOR 정답
```

```
# [3] 1000개의 샘플 랜덤 생성 (복원 추출)
N = 1000
idxs = np.random.randint(0, 4, N)  # 0~3 인덱스 중 N개 랜덤 선택
```

```
# [4] 샘플 복제
X = X_seeds[idxs]  # shape: (1000, 2)
Y = y_seeds[idxs]  # shape: (1000,)
```

```
# [5] 입력 데이터에 노이즈 추가
X += np.random.normal(scale=0.25, size=X.shape)

# 노이즈 사용 이유 : 모델이 더 일반화된 학습을 하게 되고, 단순히 "외우는 것"이 아니라 패턴을 이해하는 것에 더 가까워진다.
```

2. Shallow Neural Network (PyTorch 사용)

```
import torch.nn as nn
import torch.nn.functional as F

class shallow_neural_network(nn.Module):
    def __init__(self, num_input_features, num_hiddens):
        super().__init__()
        self.num_input_features = num_input_features
        self.num_hiddens = num_hiddens

        # [PyTorch] 레이어 정의
        self.linear1 = nn.Linear(num_input_features, num_hiddens)  # 입력 → 은닉층
        self.linear2 = nn.Linear(num_hiddens, 1)                   # 은닉층 → 출력

        # [PyTorch] 비선형 활성 함수 정의
        self.tanh = nn.Tanh()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        z1 = self.linear1(x)       # 선형 변환
        a1 = self.tanh(z1)         # 비선형 활성화
        z2 = self.linear2(a1)
        a2 = self.sigmoid(z2)      # 출력 확률화
        return a2
```

3. SGD

```
# 하이퍼파라미터 설정
num_epochs = 100
lr = 1.0
num_hiddens = 3

# 모델, 옵티마이저, 손실 함수 정의
model = shallow_neural_network(2, num_hiddens)
optimizer = optim.SGD(model.parameters(), lr=lr)
loss_fn = nn.BCELoss()  # Binary Cross Entropy Loss
```

```
# 손실 기록용 리스트
sgd_losses = []
```

```
# 학습 루프
for epoch in range(num_epochs):
    cost = 0.0

    for x, y in zip(X, Y):
        x_torch = torch.from_numpy(x)          # 입력을 텐서로 변환
        y_torch = torch.FloatTensor([y])       # 정답도 텐서로 변환

        y_hat = model(x_torch)                 # 예측 수행

        loss = loss_fn(y_hat, y_torch)         # 손실 계산
        optimizer.zero_grad()
        loss.backward()                        # 전체 배치에 대해 역전파
        optimizer.step()                       # 파라미터 업데이트

        cost += loss.item()                    # 비용 누적

    cost = cost / len(X)                       # 평균 손실

    # 손실 기록용에 추가!
    sgd_losses.append(cost)

    if epoch % 10 == 0:
        print(epoch, cost)                     # 출력
```

4. GD

```
# 하이퍼파라미터 설정
num_epochs = 100
lr = 1.0
num_hiddens = 3

# 모델, 옵티마이저, 손실 함수 정의
model = shallow_neural_network(2, num_hiddens)
optimizer = optim.SGD(model.parameters(), lr=lr)
loss_fn = nn.BCELoss()  # Binary Cross Entropy Loss
```

```
# 손실 기록용 리스트
gd_losses = []
```

```
# 학습 루프
for epoch in range(num_epochs):
    optimizer.zero_grad()

    cost = 0.0

    for x, y in zip(X, Y):
        x_torch = torch.from_numpy(x)          # 입력을 텐서로 변환
        y_torch = torch.FloatTensor([y])       # 정답도 텐서로 변환

        y_hat = model(x_torch)                 # 예측 수행

        loss_val = loss_fn(y_hat, y_torch)     # 손실 계산
        cost += loss_val                       # 비용 누적

    cost = cost / len(X)                       # 평균 손실
    cost.backward()                            # 전체 배치에 대해 역전파
    optimizer.step()                           # 파라미터 업데이트

    # 손실기록용에 추가!
    gd_losses.append(cost.item())

    if epoch % 10 == 0:
        print(epoch, cost)                     # 출력 시 .item()으로 숫자화
```

2. Compare their Loss Curves

(1) SGD

```
[10]  1 for x, y in zip(x_seeds, y_seeds):
      2     print(x) # 입력 벡터 출력
      3
      4     x_torch = torch.FloatTensor(x)      # 입력 벡터를 텐서로 변환
      5     y_hat = model(x_torch)              # 모델 예측 수행
      6
      7     print(y, y_hat.item())              # 정답 y와 예측값 ŷ 출력
      8
```

```
[0, 0.]
0 0.00017002392269205302
[1, 0.]
1 0.9834919571876526
[0, 1.]
1 0.9908343553543091
[1, 1.]
0 0.06280838698148727
```

(2) GD

```
1 for x, y in zip(x_seeds, y_seeds):
2     print(x) # 입력 벡터 출력
3
4     x_torch = torch.FloatTensor(x)      # 입력 벡터를 텐서로 변환
5     y_hat = model(x_torch)              # 모델 예측 수행
6
7     print(y, y_hat.item())              # 정답 y와 예측값 ŷ 출력
8
```

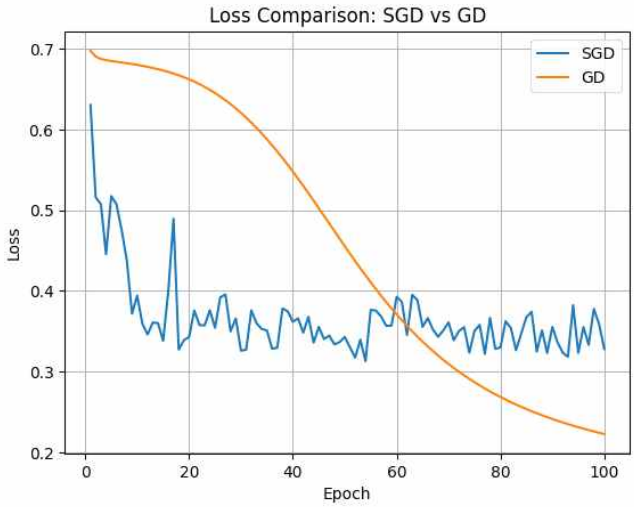
```
[0, 0.]
0 0.0294327475130558
[1, 0.]
1 0.92560213804245
[0, 1.]
1 0.9206427931785583
[1, 1.]
0 0.1318407505750656
```

5. SGD vs GD 비교 + 표 생성

```
[15]  1 # 결과를 표로 만들기
      2 df = pd.DataFrame({
      3     'Epoch': list(range(1, num_epochs + 1)),
      4     'SGD Loss': sgd_losses,
      5     'GD Loss': gd_losses
      6 })
      7
      8 print(df.head(10))  # 앞부분만 미리 보기
```

	Epoch	SGD Loss	GD Loss
0	1	0.630465	0.697641
1	2	0.516059	0.690482
2	3	0.507678	0.687571
3	4	0.445493	0.686072
4	5	0.517567	0.685025
5	6	0.507373	0.684103
6	7	0.475522	0.683192
7	8	0.438047	0.682249
8	9	0.371761	0.681247
9	10	0.394138	0.680174

```
[16]  1 # 시각화
      2 plt.plot(df['Epoch'], df['SGD Loss'], label='SGD')
      3 plt.plot(df['Epoch'], df['GD Loss'], label='GD')
      4 plt.xlabel('Epoch')
      5 plt.ylabel('Loss')
      6 plt.title('Loss Comparison: SGD vs GD')
      7 plt.legend()
      8 plt.grid(True)
      9 plt.show()
     10
```



- 위 그래프는 SGD와 GD 학습 방식에 따른 손실 변화(Loss Curve)를 비교한 결과를 보여준다.
- GD는 느리지만 안정적이고 부드러운 Loss 감소를 보여준 반면에, SGD는 초기에 빠른 Loss 감소를 보여주었지만 Loss 곡선에 진동이 많이 발생하는 모습을 보여주었습니다.
- 최종적으로 epoch가 0에서 100으로 증가하면서 GD 곡선이 더 낮은 Loss에 도달하는 모습을 볼 수 있었습니다.