CH-3

1. 다음 중 퍼셉트론의 특징으로 옳지 않은 것은?
    1. 2개 이상의 입력값을 받을 수 있다.
    2. 단층 신경망의 일종이다.
    3. 시그모이드 함수를 활성화 함수로 사용한다.
    4. 선형 분리 가능한 문제만 해결할 수 있다.
    5. 모든 입력값에 동일한 가중치를 적용한다. ❤
2. 단층 퍼셉트론으로 XOR 문제를 해결할 수 없는 이유는 무엇인가?
    1. 입력 데이터에 선형적으로 구분할 수 없는 영역이 존재하기 때문 ❤
    2. 활성화 함수로 시그모이드 함수를 사용하기 때문
    3. 은닉층이 없어 표현력이 부족하기 때문
    4. 가중치 초기화 방식이 잘못되어 있기 때문
    5. 최적화 알고리즘이 경사 하강법이 아니기 때문

Note

> - 보기 3)은 단층형태 퍼셉트론의 한계다 그러나 XOR 문제를 해결하지 못하는 문제를 원인으로 보기 어렵다.

3. 다음 중 신경망 활성함수 설명으로 옳지 않은 것은?
    1. 신경망의 *출력값 범위*를 제한한다.
    2. 신경망의 *비선형성*을 도입한다.
    3. 모든 뉴런에 동일한 활성함수를 사용해야 한다. ❤
    4. 손실 함수의 경사를 계산하는 데 사용됩니다.
    5. 경사하강법 계열의 최적화 학습을 하는 경우 미분가능해야 한다.
4. 다음 중 비선형성을 도입하지 않는 신경망 활성함수는?
    1. 시그모이드 함수
    2. ReLU 함수
    3. Tanh 함수

    4. 선형 함수 ❤
    5. softmax 함수
5. 다음 중 ReLU 함수의 단점으로 옳지 않은 것은?
    1. 계산 과정이 간단하다.
    2. 출력값이 0 또는 양의 값만 출력된다.
    3. 음의 값을 출력할 수 있다. ❤
    4. 깊은 신경망 학습에 유용하다.
    5. Dead ReLU Problem이 발생할 수 있다.
6. 소프트맥스 함수의 특징으로 옳은 것은?
    1. 입력값의 크기에 상관없이 출력값의 범위가 0과 1 사이이다.
    2. 출력값들의 합이 1이 된다.
    3. 출력층의 뉴런 개수와 무관하게 항상 유일한 최댓값을 가진다.
    4. 출력값이 클수록 해당 클래스일 확률이 높다.
    5. 위의 모든 특징을 만족한다. ❤
7. 다음 설명 중 순전파(Forward Propagation)와 역전파(Backpropagation)에 대한 설명으로 잘못된 것은?
    1. 순전파는 입력값부터 시작해 가중치와 활성화 함수를 통과하며 최종 출력값을 계산하는 과정이다.
    2. 역전파는 최종 출력값과 정답 레이블 간의 오차를 활성화 함수의 미분값으로 역전파하여 가중치를 업데이트하는 과정이다.
    3. 순전파 시에는 가중치가 업데이트되지 않는다.
    4. 역전파 시에는 연쇄법칙을 이용하여 은닉층의 가중치도 업데이트한다.
    5. 순전파와 역전파는 서로 독립적인 과정이다. ❤
8. 신경망 학습에서 손실 함수(Loss Function)의 주요 기능과 역할로 가장 적절한 것은 무엇인가?
    1. 모델의 예측 결과와 실제 값 사이의 오차를 측정하여 학습 방향을 결정한다. ❤
    2. 입력 데이터를 비선형적으로 변환하여 모델의 표현력을 향상시킨다.
    3. 학습된 모델의 복잡도를 조절하여 과적합을 방지한다.
    4. 신경망의 각 층에서 활성화 함수의 출력을 계산한다.
    5. 데이터의 분포를 추정하여 새로운 데이터를 생성한다.
9. 역전파(backpropagation)는 신경망 학습 시 가중치를 업데이트하는 방법이다. 이 과정에서 연쇄법칙(chain rule)을 적용하여 은닉층의 가중치까지 업데이트할 수 있다. 연쇄법칙은 합성함수의 미분을 간단히 계산할 수 있게 해주는 미적분의 기본 법칙이다. 역전파에서 연쇄법칙을 사용하는 이유는 무엇인가?
    1. 최종 출력의 오차를 각 가중치에 쉽게 전파하기 위해
    2. 손실 함수에 대한 각 가중치의 그래디언트를 계산하기 위해 ❤
    3. 은닉층의 활성화값을 계산하기 위해
    4. 가중치 초기값을 결정하기 위해
    5. 정규화 기법을 적용하기 위해

10. 신경망 학습률에 관한 설명으로 옳지 않은 것은?
    1. 학습률은 모델의 가중치 업데이트 크기를 조절하는 하이퍼파라미터이다.
    2. 학습률이 높으면 모델 학습 속도가 빨라지지만, 오버슈팅 (overshooting) 발생 가능성이 높아진다.
    3. 학습률이 낮으면 모델 학습 속도가 느려지지만, 안정적으로 최적점에 도달할 가능성이 높아진다.
    4. 학습률은 모델 학습 과정에서 고정되어야 한다. ❤
    5. 학습률은 모델의 성능에 영향을 미치는 중요한 요소이다.
11. 다음 중 경사 하강법(Gradient Descent)에 대한 설명으로 옳은 것은?
    1. 손실함수의 최솟값이 아닌 최댓값을 찾는 방법이다.
    2. 한 번의 반복으로 전역 최적해에 도달할 수 있다.
    3. 손실함수의 기울기와 같은 방향으로 가중치를 업데이트한다.
    4. 모든 데이터 포인트에 대해 손실을 계산한 후 가중치를 업데이트한다.
    5. 데이터 분포와 손실함수의 형태에 따라 지역 최적해에 수렴할 수 있다. ❤
12. Step, Epoch, batch에 대한 설명으로 옳지 않은 것은?
    1. Step: 모델 학습 과정에서 단일 데이터 포인트에 대한 가중치 업데이트를 의미한다.
    2. Epoch: 모델 학습 과정에서 전체 학습 데이터를 한 번 반복하는 것을 의미한다.
    3. batch: 모델 학습 과정에서 전체 학습 데이터를 사용하여 가중치를 업데이트한다. ❤
    4. batch: 모델 학습 속도를 향상시키기 위해 사용된다.
    5. batch: 학습 데이터의 특성에 따라 적절한 크기를 선택해야 한다.
13. Local Minima(지역 최소점)는 경사 하강법과 같은 반복 최적화 알고리즘에서 발생할 수 있는 문제 중 하나이다. 지역 최소점에 수렴하게 되면 전역 최적해를 찾는 데 어려움이 있다. 다음 설명 중 지역 최소점(Local Minima)에 대한 설명으로 옳은 것은?
    1. 전역 최소점에 해당하므로 목표 지점이다.
    2. 손실함수의 기울기가 0이 아닌 지점이다.
    3. 주변 영역의 값보다 작지만 전체에서 최솟값은 아니다. ❤
    4. 초기 가중치 값과 무관하게 발생한다.
    5. 경사 하강법에서만 발생하는 문제이다.
14. 다음 중 신경망 최적화 기법으로 올바르지 않은 것은?
    1. 경사하강법: 손실 함수의 최솟값을 찾는 방향으로 가중치를 업데이트하는 알고리즘
    2. Adam: 경사 하강법의 변형으로, 학습률 조절을 자동화하여 안정적인 학습을 가능하게 하는 알고리즘
    3. mini-batch SGD: 전체 학습 데이터 사용하여 가중치를 업데이트하는 알고리즘 ❤

4. Momentum: 이전 업데이트 방향을 고려하여 현재 업데이트 방향을 조절하는 알고리즘
5. Adagrad: 과거 업데이트 크기에 따라 학습률을 조절하여, 각 가중치의 업데이트 속도를 개별적으로 조절하는 알고리즘

torch - SGD optimize

- SGD는 각 업데이트 단계(step)에서 하나의 무작위로 선택된 데이터 포인트에 대해서만 기울기를 계산하는 확률적 방법을 의미한다.
- 그러나 pytorch의 SGD 는 실제로는 Mini-batch Gradient Descent를 의미한다.

15. PyTorch에서 선형 신경망 모델을 학습할 때, 각 학습 단계(step)에서 `optimizer.zero_grad()`를 호출하는 주된 이유는 무엇인가?
    1. 모델의 가중치를 초기화하기 위해
    2. 손실 함수의 값을 0으로 설정하기 위해
    3. 이전 학습 단계에서 계산된 그래디언트(gradients)를 초기화하기 위해 ❤
    4. 학습률(learning rate)을 0으로 설정하기 위해
    5. 모델의 예측값을 0으로 설정하기 위해
16. PyTorch에서 신경망 모델을 학습할 때, 다음 중 `.backward()`와 `.step()` 메서드의 역할에 대한 설명으로 올바른 것은 무엇인가?
    1. `.backward()`는 모델의 가중치를 업데이트하고, `.step()`은 손실 함수를 계산.
    2. `.backward()`는 손실 함수의 그래디언트를 계산하고, `.step()`은 계산된 그래디언트를 기반으로 가중치를 업데이트. ❤
    3. `.backward()`는 모델의 예측값을 계산하고, `.step()`은 학습률을 조정.
    4. `.backward()`는 모델의 구조를 정의하고, `.step()`은 모델의 초기 가중치를 설정.
    5. `.backward()`는 학습 데이터를 모델에 입력하고, `.step()`은 모델의 성능을 평가.
17. PyTorch의 `torch.utils.data.Dataset` 클래스는 사용자 정의 데이터셋을 생성하기 위한 추상 클래스이다. 다음 중 `Dataset` 클래스를 상속받아 사용자 정의 데이터셋을 만들 때 반드시 구현해야 하는 메서드는 무엇인가?
    1. `__init__()`와 `__load__()`
    2. `__len__()`과 `__getitem__()` ❤
    3. `__batch__()`와 `__shuffle__()`
    4. `__transform__()`과 `__collate__()`
    5. `__save__()`와 `__export__()`

torch - torch.utils.data.Dataset

- **__len__() :**
  - 데이터셋의 크기(샘플 수)를 반환하는 메서드.
  - `len(dataset)`과 같이 데이터셋의 길이를 얻기 위해 사용.
- **__getitem__(idx) :**
  - 주어진 인덱스 `idx`에 해당하는 샘플을 반환하는 메서드.
  - `dataset[idx]`와 같이 데이터셋의 특정 샘플에 접근하기 위해 사용.
- `Dataset` 클래스를 상속받아 사용자 정의 데이터셋을 만들 때, 이 두 메서드를 반드시 구현해야 한다.
- `DataLoader`와 같은 데이터 로더는 이 메서드들을 사용하여 데이터셋의 샘플에 접근하고 배치(batch)를 생성.

18. PyTorch의 `torch.utils.data.DataLoader`는 데이터셋을 배치 단위로 묶어 모델 학습에 효율적으로 제공하는 역할을 한다. 다음 중 `DataLoader`의 주요 기능으로 올바르지 않은 것은 무엇인가?
    1. 데이터셋의 샘플을 무작위로 섞어(shuffle) 학습의 일반화 성능을 향상시킨다.
    2. 데이터셋의 샘플을 배치(batch) 단위로 묶어 메모리 효율성을 높인다.
    3. 데이터셋의 샘플을 병렬로 로드하여(num_workers) 데이터 로딩 속도를 향상시킨다.
    4. 데이터셋의 샘플을 모델의 입력 형태에 맞게 변환(transform)한다. ❤
    5. 데이터셋의 샘플을 GPU 메모리에 미리 로드하여(pin_memory) 학습 속도를 향상시킨다.

torch - troch.utils.data.DataLoader
- 이미지의 경우는 torchvision.transforms를 사용하여 형태을 변경한다.
- 사용자 정의 형태 변경을 하기 위해서는 Dataset 클래스에서 사용자 정의 변환 함수를 사용해야 한다.

19. 다음 중 신경망 학습 과정에서 발생하는 단계를 순서대로 올바르게 나열한 것은 무엇인가요?
    1. 데이터 입력 -> 손실 계산 -> 순전파 -> 역전파 -> 가중치 업데이트 -> 활성화 함수 적용
    2. 데이터 입력 -> 활성화 함수 적용 -> 순전파 -> 손실 계산 -> 역전파 -> 가중치 업데이트
    3. 데이터 입력 -> 순전파 -> 활성화 함수 적용 -> 손실 계산 -> 역전파 -> 가중치 업데이트 ❤
    4. 데이터 입력 -> 역전파 -> 손실 계산 -> 순전파 -> 활성화 함수 적용 -> 가중치 업데이트
    5. 데이터 입력 -> 가중치 업데이트 -> 순전파 -> 활성화 함수 적용 -> 손실 계산 -> 역전파

20. PyTorch를 사용하여 신경망 모델을 학습하는 코드 조각이다. 다음 중 주석으로 ???? 처리된 부분에 들어갈 코드로 올바른 것은 무엇인가? (데이터는 load된 뒤라 전제한다.)

```
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
from torch import nn
import torch.nn.functional as F
from torch import optim

# 하이퍼파라메터 설정
epochs = 100
batch = 64
lr = 0.3

# 훈련데이터를 위한
train_ds = TensorDataset(x_train, y_train)
train_dl = DataLoader(train_ds, batch_size=batch)

# 모델 정의
class Mnist_Logistic(nn.Module):
    def __init__(self):
        super().__init__()
        self.lin_01 = nn.Linear(784,10)
        self.sig = nn.Sigmoid()

    def forward(self, xb):
        x_out = self.lin_01(xb)
        out = self.sig(x_out)
        return out
model = Mnist_Logistic()

# 최적화 기법과 학습률 지정
optimizer = optim.SGD(model.parameters(), lr=0.01)

# 손실함수 지정
criterion = F.cross_entropy()

# 학습
for epoch in range(epochs):
    for xb, yb in train_dl:
        out = model(xb)
        # 손실 계산
        loss = criterion(out, yb)

        # 역전파
        # ????

        # 가중치 업데이트
```

```
        optimizer.step()
        # 그래디언트 초기화
        optimizer.zero_grad()
```

1) `loss.backward()` ❤
2) `optimizer.backward()`
3) `model.backward()`
4) `inputs.backward()`
5) `outputs.backward()`

> **Note**
>
> - **loss.backward():**
>   - `loss.backward()` 는 손실 함수(loss)에 대한 각 매개변수(parameter)의 그래디언트(gradient)를 계산.
>   - 계산된 그래디언트는 각 매개변수의 `.grad` 속성에 저장되며, `optimizer.step()` 을 통해 가중치를 업데이트하는 데 사용.
> - **optimizer.step():**
>   - 옵티마이저(optimizer)를 사용하여 계산된 그래디언트를 기반으로 모델의 가중치를 업데이트.
> - **optimizer.zero_grad():**
>   - 이전 학습 단계에서 계산된 그래디언트를 초기화.

21. PyTorch에서 모델을 평가(evaluation)할 때, `torch.eval()` 과 `with torch.no_grad()` 를 함께 사용하는 이유는 무엇인가요?

```python
# 평가 모드 설정 및 추론
model.eval()
with torch.no_grad():
    predict = model(inputs)
    print("예측 결과:", predict)
    loss = criterion(predict, label)
    print("평가 손실:", predict)

# 모델의 학습 모드 확인
print("모델 모드:", model.training)
```

- 1. 모델의 가중치를 초기화하고 학습률을 조정하기 위해
  2. 모델의 예측값을 계산하고 손실 함수를 평가하기 위해
  3. 모델의 학습 모드를 평가 모드로 변경하고 그래디언트 계산을 비활성화하여 메모리 사용량을 줄이고 추론 속도를 높이기 위해 ❤

4. 모델의 구조를 정의하고 입력 데이터의 형태를 변환하기 위해
5. 모델의 성능을 시각화하고 학습 곡선을 분석하기 위해

Note

- **`torch.eval()` :**
  - 모델을 평가 모드(evaluation mode)로 전환.
  - 평가 모드에서는 드롭아웃(dropout)이나 배치 정규화(batch normalization)와 같은 학습 시에만 사용되는 레이어들이 평가에 적합하게 동작.
- **`with torch.no_grad()` :**
  - 그래디언트 계산을 비활성화.
  - 평가 시에는 가중치를 업데이트할 필요가 없으므로 그래디언트 계산을 하지 않아 메모리 사용량을 줄이고 추론 속도를 높일 수 있다.
  - 모델이 학습이 아닌 **추론(inference) 과정**에 있을 때 필수적으로 사용.