4장. 딥러닝 시작

인공 신경망에서 이용하는 구조는 퍼셉트론

- AND 게이트 : 모든 입력이 1일 때 작동
- OR 게이트 : 입력에서 둘 중 하나만 1이거나 모두 1일 때 작동
- XOR 게이트 : 두개 중 한개만 1일 때 작동 → 데이터가 비선형적으로 분리되어 제대로 된 분류가 어려움

→단층 퍼셉트론에서는 AND, OR 연산에 대해서는 학습이 가능하지만 XOR에 대해서는 학습이 불가능!

⇒ 입력층과 출력층 사이에 하나 이상의 중간층(은닉층)을 두어, 비선형적으로 분리되는 데이터에 대해 학습 가능 (다층 퍼셉트론)

> 💡 심층 신경망(DNN, 딥러닝) : 입력층과 출력층 사이에 은닉층이 여러개 있는 신경망

딥러닝 용어

- 가중치: 입력 값이 연산 결과에 미치는 영향력을 조절하는 요소
- 가중합, 전달함수: 각 노드에서 들어오는 신호에 가중치를 곱해 다음 노드로 전달될 때, 이 값들을 모두 더한 합계
- 활성화 함수: 전달 함수에서 전달받은 값을 출력할 때 일정 기준에 따라 출력 값을 변화시키는 비선형 함수
    - 시그모이드 함수: 선형 함수의 결과를 0~1 사이에서 비선형 형태로 변형, 딥러닝 모델의 깊이가 깊어지면 기울기 소멸 문제가 발생함, 결과값의 평균이 0이 아닌 양수로 편향됨
    - 하이퍼볼릭 탄젠트 함수: 선형 함수의 결과를 -1~1 사이에서 비선형 형태로 변형, 결과값의 평균이 0이 아닌 양수로 편향된 문제를 해결했으나 기울기 소멸 문제 여

전히 있음

- ReLU 함수: 입력값(x)이 음수일 때는 0을 출력하고 양수일 때는 x를 출력, 학습 속도가 빠르고 기울기 소멸 문제가 발생하지 않음 but 음수 값을 입력 받으면 항상 0을 출력하기 때문에 학습 능력 감소

  ⇒ Ricky ReLU 함수 사용

- Ricky ReLU 함수: 입력 값이 음수이면 0이 아닌 매우 작은 수 반환 → 입력 값이 수렴하는 구간이 제거되어 ReLU 함수의 문제 해결

- Softmax 함수: 입력 값을 0~1 사이에 출력되도록 정규화하여 출력 값들의 총합이 항상 1이 되도록 함, 주로 출력 노드의 활성화 함수로 사용됨

```
class Net(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(Net, self).__init__()
        self.hidden = torch.nn.Linear(n_feature, n_hidden) #은닉층
        self.relu = torch.nn.ReLu(inplace=True)
        self.out = torch.nn.Linear(n_hidden, n_output)
        self.softmax = torch.nn.Softmax(dim=n_output)
    def forward(self, x):
        x = self.hidden(x)
        x = self.relu(x) #은닉층을 위한 ReLu 활성화 함수
        x = self.out(x)
        x = self.softmax(x) #출력층을 위한 softmax 활성화 함수
        return x
```

- 손실 함수: 학습을 통해 얻은 데이터의 추정치가 실제 데이터와 얼마나 차이가 나는지 평가하는 지표, 값이 클수록 많이 틀리고, 0에 가까울수록 와나벽하게 추정할 수 있다는 의미

  - 평균 제곱 오차(MSE): 회귀에서 손실 함수로 주로 사용됨

```
loss_fn = torch.nn.MSELoss(reduction='sum')
y_pred = model(x)
loss = loss_fn(y_pred, y)
```

  - 크로스 엔트로피 오차(CEE): 분류 문제에서 원-핫 인코딩 했을 때만 사용할 수 있는 오차 계산법

    - 일반적으로 분류에서는 시그모이드 함수 사용 → 자연 상수 때문에 MSE를 적용하면 매끄럽지 못한 그래프 출력, local minimum에서 학습이 멈출 수 있음 → 자연 로그를 취한 크로스 엔트로피 사용

```
loss = nn.CrossEntropyLoss()
input = torch.randn(5, 6, requires_grad = True)
target = torch.empty(3, dtype=torch.long).random_(5)
output = loss(input, target)
output.backward()
```