4장 딥러닝 시작

날짜	@2025년 3월 12일 → 2025년 3월 17일
진행 상태	진행 중

[목차]

4.1 인공 신경망의 한계와 딥러닝 출현

💡 **퍼셉트론**
: 흐름이 있는
다수의 신호를 입력받아 **하나의 신호를 출력**하는데, 이 신호를 입력을 받아 '흐른다/안 흐른다(1 or 0)'는 정보를 앞으로 전달하는 원리로 작동

- AND 게이트: 모든 입력이 '1' 일 때 작동

- OR 게이트: 입력 중 둘 중 하나면 '1' 이거나 둘다 '1'일 때 작동
- XOR 게이트: 입력 중 한 개만 '1' 일 때 작동

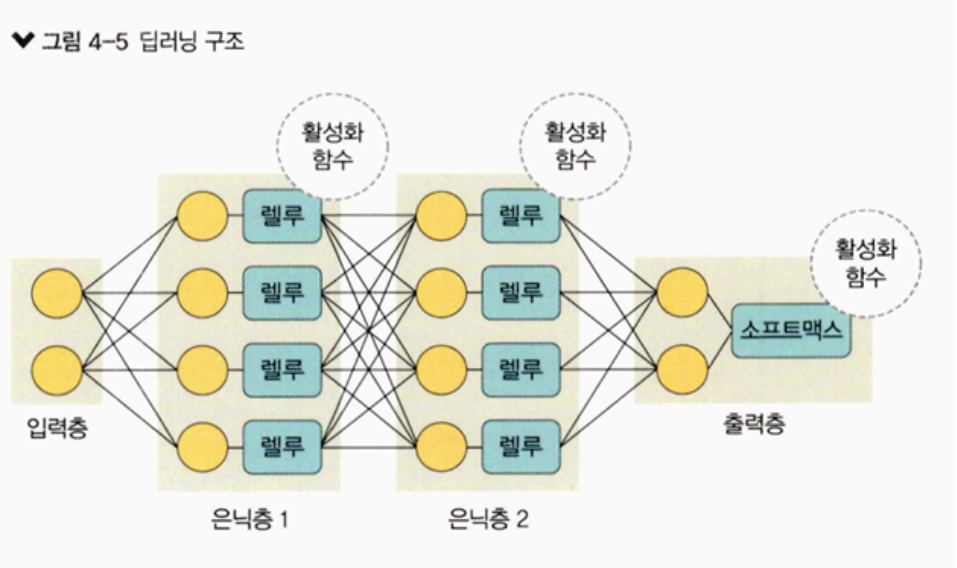
⇒ XOR 게이트는 데이터가 비선형적으로 분리되어 제대로 된 분류가 어려움 → AND, OR 연산에서는 학습이 가능, XOR 학습 불가

⇒ 이를 극복 : 입력층과 출력층 사이의 **중간층(은닉층)**

> 심층 신경망: 입력층과 출력층 사이에 은닉층이 여러 개 있는 신경망 = 딥러닝

4.2 딥러닝 구조

4.2.1 딥러닝 용어



[가중치]

입력 값이 연산 결과에 미치는 영향력을 조절하는 요소

[가중합(=전달 함수)]

각 노드에서 들어오는 신호에 가중치를 곱해 다음 노드로 전달하는데, 이 값들을 모두 더한 합계를 가중합이라고 한다.

또한, 노드의 가중합이 계산되면 이 가중합을 활성화 함수로 보내기 때문에 전달함수(transfer function)이라고도 함

$$\sum_i (w_i x_i + b)$$

- 입력값 x_i, 가중치 w_i

[활성화 함수]

시그모이드 함수(sigmoid)

- 선형 함수의 결과를 0~1 사이에서 비선형 형태로 변형
- 주로 로지스틱 회귀와 같은 분류 문제를 확률적으로 표현하는데 사용
- '기울기 소멸 문제(vanishing gradient problem)'가 발생하여 딥러닝 모델에서 잘 사용하지 않는 추세
- f(X) = 1 / (1+e^(-x))

하이퍼볼릭 탄젠트 함수(hyperbolic tangent)

- 선형 함수의 결과를 -1~1 사이에서 비선형 형태로 변형
- 시그모이드에서 결괏값의 평균이 0이 아닌 양수로 편향된 문제를 해결하는데 사용했지만
- 여전히 기울기 소멸 문제 발생

렐루 함수(ReLU)

- 최근에 활발히 사용됨
- 입력(x)이 음수일 때는 0출력, 양수일 때는 x 출력
- 경사하강법에 영향을 주지 X → 빠른 학습속도, 기울기 소멸 문제 X
- 일반적으로 은닉층에서 사용
- 하이퍼볼릭 탄젠트 함수 대비 학습 속도가 6배 빠름
- 음수값을 받으면 항상 0을 출력해 학습 능력이 감소 하는 문제가 있어 리키 렐루 함수 등을 사용

리키 렐루 함수(Leaky RelU)

- 입력 값이 음수이면 0이 아닌 0.001처럼 매우 작은 수를 반환 → 입력 값이 수렴하는 구간이 제거됨(렐루 문제 해결)

소프트맥스 함수(softmax)

- 입력 값을 0~1 사이에 출력되도록 정규화하여 출력 값들의 총합이 항상 1이 되도록 함
- 보통 딥러닝에서 출력 노드의 활성화 함수로 많이 사용

```
## 렐루 함수와 소프트맥스 함수 구현하기

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

[손실 함수]

경사 하강법은 학습률과 손실 함수의 순간 기울기를 이용하여 가중치를 업데이트하는 방법

평균제곱오차(Mean Squared Error, MSE)

- MSE: 실제 값과 예측 값의 차이를 제곱하여 평균
- 따라서 값이 작을수록 예측력이 좋음
- 회귀에서 손실함수로 주로 사용됨

```
import torch

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

크로스 엔트로피 오차(CrossEntropy Error)

- **분류문제**에서 **원-핫 인코딩** 했을 때만 사용할 수 있음
- 일반적으로 분류문제에서는 데이터의 출력을 0과 1로 구분하기 위해 시그모이드 함수를 사용 → 시그모이드 함수에 포함된 자연 상수 e 때문에 평균 제곱 오차를 적용하면 매끄럽지 못한 그래프가 출력됨. 따라서 크로스 엔트로피 사용
- 경사 하강법에서 학습이 지역 최소점에서 멈출 수 있음 ⇒ 자연로그를 모델의 출력값에 취함

```
loss = nn.CrossEntropyL
input = torch.randn(5,6, r
target = torch.empty(3, c
output.backward()
```

$$CrossEntropy = -\sum_{i=1}^{n} y_i \log \hat{y}_i$$

$\hat{y}_i$: 신경망의 출력(신경망이 추정한 값)
y_i: 정답 레이블
i: 데이터의 차원 개수

4.2.2 딥러닝 학습

순전파(feedforward)

: 네트워크에 훈련 데이터가 들어올 때 발생, 데이터를 기반으로 예측 값을 계산하기 위해 전체 신경망을 교차해 지나감

- 모든 뉴런이 이전 층의 뉴런에서 수신한 정보에 변환(가중합 및 활성함수)을 적용하여
- 다음층 (은닉층)의 뉴런으로 전송
- 모든 뉴런 계산 완료하면 출력층에 도달

역전파(backpropagation)

⚙️ 손실함수 :네트워크의 예측 값과 실제 값의 차이(손실, 오차)를 추정

- 손실 함수 비용은 '0'이 이상적
- '0'에 가깝게 하기 위해 훈련을 반복하며 가중치를 조정

: 손실(오차)가 계산되면 그 정보는 (출력층→은닉층→입력층) 순으로 전파됨

- 손실 비용은 은닉층의 모든 뉴런으로 전파됨
- 각 뉴런이 원래 출력에 기여한 가중치에 따라 값이 달라짐

  = 기존의 가중치 - [(예측값-실제값)을 각 뉴런의 가중치로 미분]

  → 각 값을 순전파의 가중치 값으로 사용

4.2.3. 딥러닝의 문제점과 해결 방안

💡 딥러닝의 핵심 : 활성화 함수가 적용된 여러 은닉층을 결합하여 비선형 영역을 표현

과적합 문제(over-fitting)

: 훈련 데이터를 과하게 학습할 때 발생

- 일반적으로 훈련 데이터는 실제 데이터의 일부분
- 훈련데이터 과잉 학습 → (예측값-실제값)인 오차가 감소 ↔ 검증 데이터에 대한 오차 증가
- 과소적합(under-fitting) / 적정적합(generalized-fitting) / 과적합(over-fitting)

드롭아웃(dropout)

: 신경망 모델이 과적합되는 것을 피하기 위해 임의로 일부 노드들의 학습을 제외시킴

```
class DropoutModel(torch.nn.Module):
    def __init__(self):
```

```
        super(DropoutModel, self).__init__()
        self.layer1 = torch.nn.Linear(784,1200)
        # 50% 노드를 무작위로 선택해 사용하지 않겠다는 의미
        self.dropout1 = torch.nn.Dropout(0.5)
        self.layer2 = torch.nn.Linear(1200,1200)
        self.dropout2 = torch.nn.Dropout(0.5)
        self.layer3 = torch.nn.Linear(1200, 10)
    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = self.dropout1(x)
        x = F.relu(self.layer2(x))
        x = self.dropout2(x)
        return self.layer3(x)
```

기울기 소멸 문제

: 은닉층이 많은 신경망에서 주로 발생

출력층에서 은닉층으로 전달되는 오차가 크게 줄어들어 학습이 되지 않는 현상

→ 기울기가 소멸되기 때문에 학습되는 양이 '0'에 가까워져 학습이 더디게 진행되다 오차를 줄이지 못하고 그 상태로 수렴하는 현상

배치 경사 하강법(Batch Gradient Descent, BGD)

: 전체 데이터셋에 대한 오류를 구한 후, 기울기를 한 번만 계산하여 모델의 파라미터를 업데이트

- 전체 훈련 데이터셋에 대해 가중치를 편미분
- 한 스텝에 모든 훈련 데이터셋을 사용해서 학습이 오래 걸림

확률적 경사 하강법(Statistic Gradient Descent, SGD)

: 임의로 선택한 데이터에 대해 기울기를 계산

- 적은 데이터를 사용 → 빠른 계산 가능
- 파라미터 변경 폭이 불안정하고, 때로 정확도가 낮을 수 있지미나 속도가 빠르다는 장점

미니 배치 경사 하강법(mini-batch gradient descent)

: 전체 데이터셋을 미니 배치 여러개로 나눔

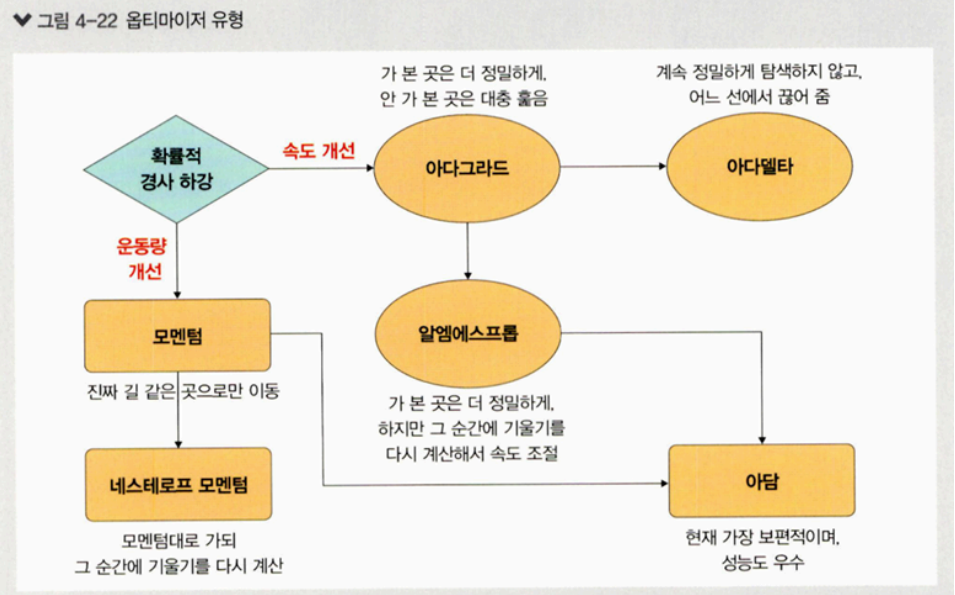
- 미니 배치마다 기울기를 구함

- 그것의 평균 기울기를 이용해 모델을 업데이트해서 학습
- 전체 데이터 계산보다 빠르고, 확률적 경사 하강법보다 안정적

```
class CustomDataset(Dataset):
    def __init__(self):
        self.x_data = [[1,2,3]],[4,5,6],[7,8,9]]
        self.y_data = [[12],[18],[11]]
        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()
    dataloader = DataLoader(
        dataset,
        batch_size=2,
        shuffle=True
    )
```

(NOTE) 옵티마이저 Optimizer

: 확률적 경사 하강법의 파라미터 변경 폭이 불안정한 문제를 해결하기 위해 학습 속도와 운동량을 조절하는 옵티마이저를 적용할 수 있다



4.2.4 딥러닝을 사용할 때 이점

특성 추출

: 데이터별로 어떤 특징을 가지고 있는지 찾아내고, 그것을 토대로 데이터를 벡터로 변환하는 작업

- 딥러닝에서는 특성 추출 과정을 알고리즘에 통합시킴
- 데이터 특성을 잡아내고자 은닉층을 깊게 쌓는 방식으로 파라미터를 늘린 모델 구조 덕분

빅데이터의 효율적 사용

딥러닝 학습을 이용한 특성 추출은 데이터 사례가 많을수록 성능이 향상됨.

4.3 딥러닝 알고리즘

4.3.1 심층 신경망(DNN)

: 입력층과 출력층 사이에 다수의 은닉층을 포함하는 인공 신경망

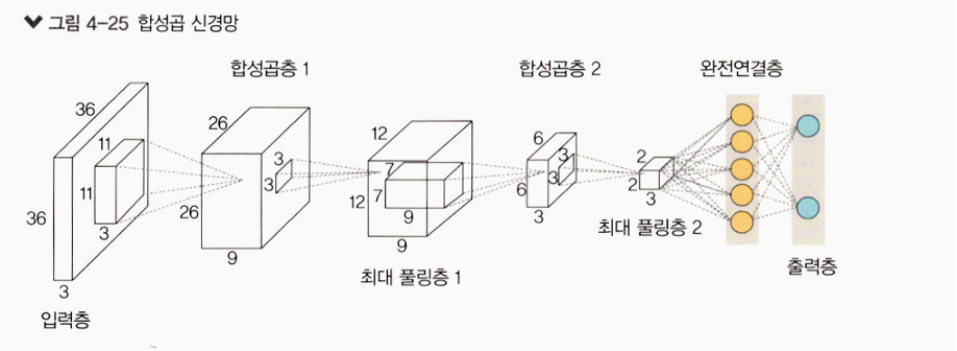
- 다양한 비선형적 관계를 학습할 수 있음

- 연산량이 많고, 기울기 소멸 문제 등 발생
- 이를 위해 드롭아웃, 렐루 함수, 배치 정규화 등 필요

4.3.2 합성곱 신경망(CNN)

: 합성곱층(convolutional layer)과 풀링층(pooling layer)을 포함하는 이미지 처리 성능이 좋은 인공 신경망 알고리즘

- 영상 및 사진이 포함된 이미지 데이터에서 객체 탐색, 위치 탐지에 유용함
- 이미지에서 객체, 얼굴, 장명을 인식하기 위해 패턴을 찾는 데 유용함



대표적인 CNN: LeNEt-5, AlexNet

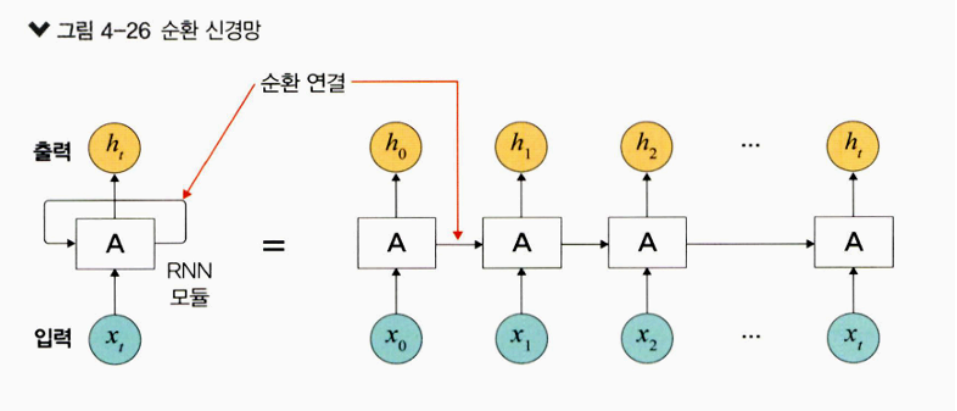
층이 더 깊은 신경망: VGG, GoogLeNet, ResNet

- 각 층의 입출력 형상을 유지
- 이미지의 공간 정보를 유지하면서 인접 이미지와 차이가 있는 특징을 효과적으로 인식
- 복수 필터로 이미지의 특성을 추출하고 학습
- 추출한 이미지의 특징을 모으고 강화하는 풀링층
- 필터를 공유 파라미터로 사용하기 때문에 일반 인공 신경망과 비교하여 학습 파라미터가 매우 적음

4.3.3 순환 신경망(RNN)

: 시계열 데이터(음악, 영상 등) 같은 시간 흐름에 따라 변화하는 데이터를 학습하기 위한 인공 신경망

- '순환': 자기 자신을 참조한다는 뜻, 현재 결과가 이전 결과와 연관 있다는 의미



- 시간성(temporal property)을 가진 데이터가 많음
- 시간성 정보를 이용하여 데이터 특징을 잘 다룸
- 시간에 따라 내용이 변하므로 데이터는 동적, 길이가 가변적
- 매우 긴 데이터를 처리하는 연구가 활발히 진행됨

- 순환 신경망은 기울기 소멸문제로 학습이 제대로 되지 않는 문제가 있음

→ 이를 해결하고자 메모리 개념을 도입한 LSTM이 많이 사용됨

4.3.4. 제한된 볼츠신 머신

: 가시층(visible layer)와 은닉층(hidden layer)로 구성되는데

가시층은 은닉층과만 연결된 모델

가시층과 가시층, 은닉층과 은닉층 사이 연결 X

- 차원 감소, 분류, 선형 회귀 분석, 협업 필터링, 특성 값 학습, 주제 모델링
- 기울기 소멸 문제를 해결하기 위해 사전 학습 용도로 활용 가능
- 심층 신뢰 신경망(DBN)의 요소로 활용됨

4.3.5 심층 신뢰 신경망

: 입력층과 은닉층으로 구성된 볼츠만 머신을 블록처럼 여러 층으로 쌓은 형태로 연결된 신경망

→ 사전 훈련된 제한된 볼츠만 머신을 층층히 쌓아 올린 구조로, 레이블이 없는 데이터에 대한 비지도 학습이 가능

[학습 절차]

1. 가시층과 은닉층 1에 제한된 볼츠만 머신을 사전 훈련
2. 첫 번째 층 입력 데이터와 파라미터를 고정하여 두 번째 층 제한된 볼츠만 머신을 사전 훈련
3. 원하는 층 개수만큼 제한된 볼츠만 머신을 쌓아 올려 전체 DBN을 완성

- 순차적으로 심층 신뢰 신경망을 학습시켜 계층적 구조를 생성
- 비지도 학습으로 학습
- 위로 올라갈수록 추상적 특성을 추출
- 학습된 가중치를 다중 퍼셉트론의 가중치 초깃값으로 사용