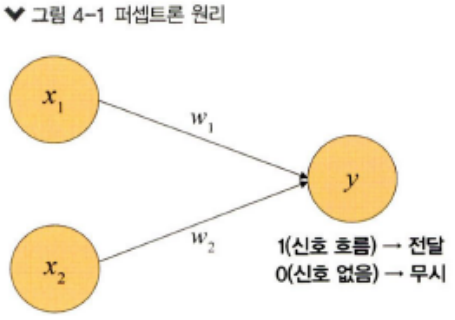
[딥러닝 파이토치 교과서] 4장 딥러닝 시작(4. 1-4.2)

작성일시	@2024년 9월 29일 오후 12:02
분야	DL
주제	기초
type	필사
날짜	@2024년 9월 28일

4.1 인공 신경망의 한계와 딥러닝 출현

퍼셉트론 : 선형 분류기. 오늘날 신경망(딥러닝)의 기원이 되는 알고리즘.

다수의, 흐름이 있는 신호를 입력으로 받아 하나의 신호를 출력하는데, 이 신호를 입력으로 받아 '흐른다(1)/안 흐른다(0)'는 정보를 앞으로 전달하는 원리로 작동한다.



입력이 두 개 있다고 할 때 컴퓨터가 논리적으로 인식하는 방식을 알아보자.

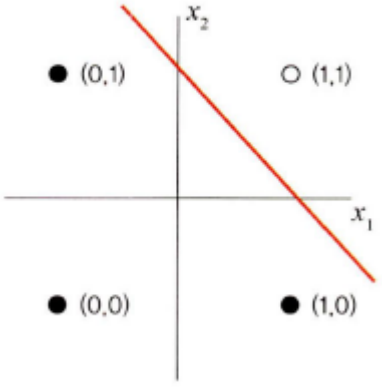
1. **AND 게이트**

   모든 입력이 1일 때 작동한다. 즉, 입력 중 어떤 하나라도 0을 가지면 작동을 멈춘다.

❯ 표 4-1 AND 게이트

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

❯ 그림 4-2 AND 게이트



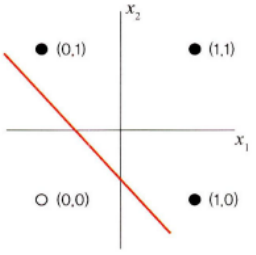
2. **OR 게이트**

   입력에서 둘 중 하나만 1이거나 둘 다 1일 때 작동한다. 즉, 입력 모두가 0을 갖는 경우를 제외한 나머지가 모두 1 값을 갖는다.

❯ 표 4-2 OR 게이트

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

❯ 그림 4-3 OR 게이트



3. **XOR 게이트**

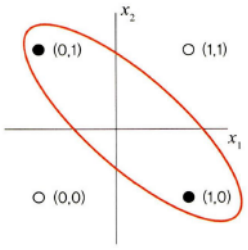
   XOR 게이트는 배타적 논리합이라는 용어로 입력 두 개 중 한 개만 1일 때 작동하는 논리 연산이다.

❯ 표 4-3 XOR 게이트

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

XOR 게이트는 데이터가 비선형적으로 분리되기 때문에 제대로 된 분류가 어렵다. 즉, 단층 퍼셉트론에서는 AND, OR 연산에 대해서는 학습할 수 있지만, XOR에 대해서는 학습이 불가능하다.

❯ 그림 4-4 XOR 게이트



→ 극복 방안 : 다층 퍼셉트론(multi-layer perceptron)

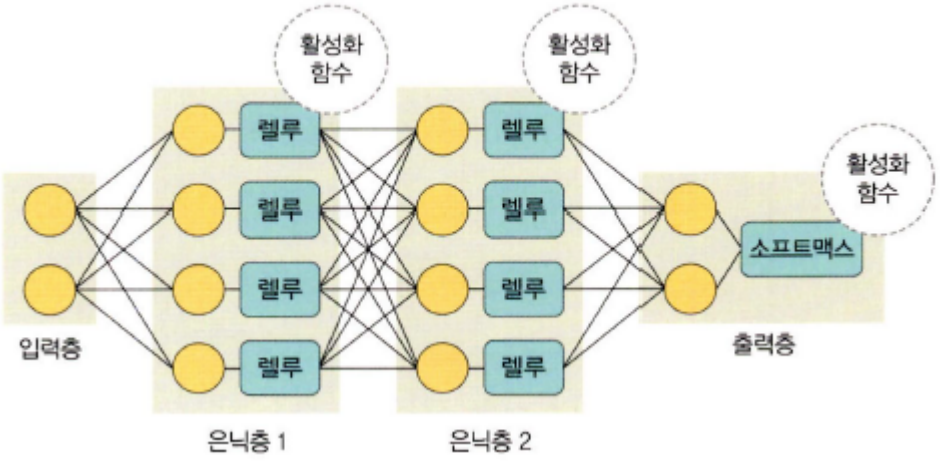
입력층과 출력층 사이에 하나 이상의 중간층(은닉층)을 두어 비선형적으로 분리되는 데이터에 대해서도 학습할 수 있도록 한다.

→ 심층 신경망(Deep Neural Network, DNN, 딥러닝) : 입력층과 출력층 사이에 은닉층이 여러 개 있는 신경망

4.2 딥러닝 구조

4.2.1 딥러닝 용어

❤ 그림 4-5 딥러닝 구조



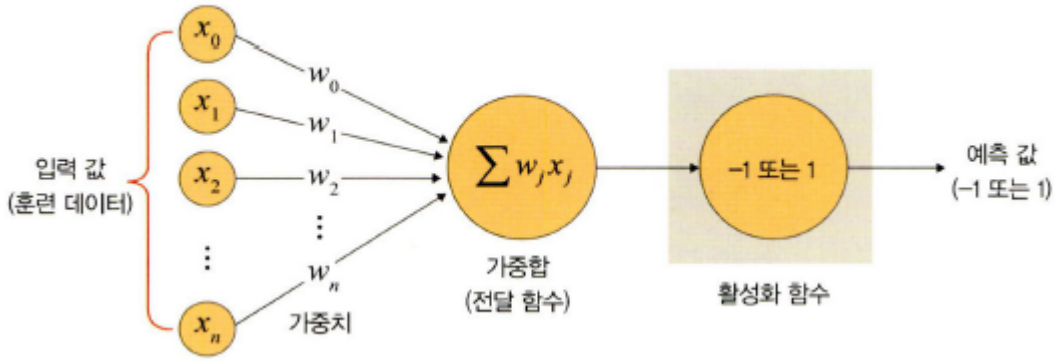
❤ 표 4-4 딥러닝 구성 요소

구분	구성 요소	설명
층	입력층(input layer)	데이터를 받아들이는 층
	은닉층(hidden layer)	모든 입력 노드부터 입력 값을 받아 가중합을 계산하고, 이 값을 활성화 함수에 적용하여 출력층에 전달하는 층
	출력층(output layer)	신경망의 최종 결괏값이 포함된 층

구분	구성 요소	설명
가중치(weight)		노드와 노드 간 연결 강도
바이어스(bias)		가중합에 더해 주는 상수로, 하나의 뉴런에서 활성화 함수를 거쳐 최종적으로 출력되는 값을 조절하는 역할을 함
가중합(weighted sum), 전달 함수		가중치와 신호의 곱을 합한 것
함수	활성화 함수(activation function)	신호를 입력받아 이를 적절히 처리하여 출력해 주는 함수
	손실 함수(loss function)	가중치 학습을 위해 출력 함수의 결과와 실제 값 간의 오차를 측정하는 함수

1. **가중치** : 입력 값이 연산 결과에 미치는 영향력을 조절하는 요소.

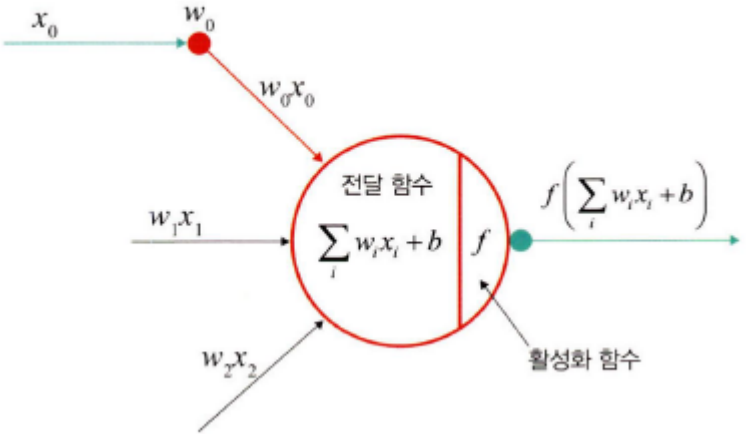
❯ 그림 4-6 가중치



w_1 값이 0 혹은 0.001이라면, x_1이 아무리 큰 값이라도 $x_1 \times w_1$ 값은 0이거나 0에 가까운 값이 된다.

2. **가중합(전달 함수)** : 각 노드에서 들어오는 신호에 가중치를 곱한 값을 모두 더한 합계. 노드의 가중합이 계산되면 이 가중합을 활성화 함수로 보내기 때문에 '전달' 함수라고도 한다.

❯ 그림 4-7 전달 함수



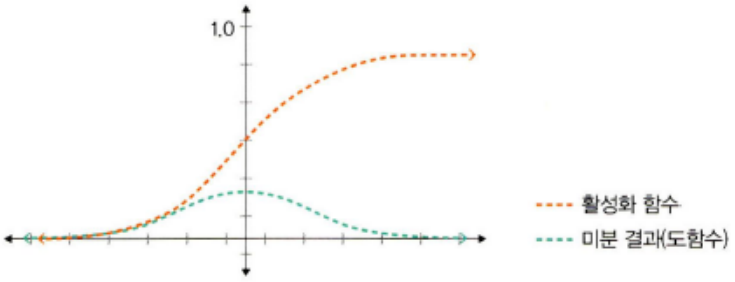
$$\sum_i w_i x_i + b$$

(w: 가중치, b: 바이어스)

3. **활성화 함수** : 전달 함수에서 전달받은 값을 출력할 때 일정 기준에 따라 출력 값을 변화시키는 비선형 함수.
    a. **시그모이드 함수** : 선형 함수의 결과를 0에서 1 사이에서 비선형 형태로 변형해 준다. 주로 로지스틱 회귀와 같은 분류 문제를 확률적으로 표현하는 데 사용된다. 딥러닝 모델의 깊이가 깊어지면 기울기가 사라지는 '기울기 소멸 문제'가 발생하여 딥러닝 모델에서는 잘 사용하지 않는다.

$$f(x) = \frac{1}{1+e^{-x}}$$

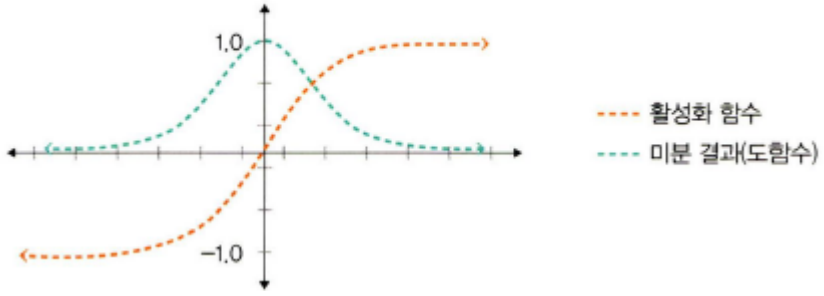
❯ 그림 4-8 시그모이드 활성화 함수와 미분 결과



- 비선형 함수 : 직선으로 표현할 수 없는 데이터 사이의 관계를 표현하는 함수

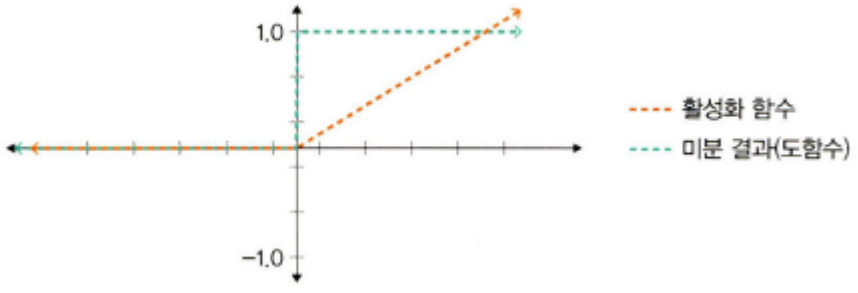
b. **하이퍼볼릭 탄젠트** : 선형 함수의 결과를 -1에서 1 사이에서 비선형 형태로 변형해 준다. 시그모이드에서 결괏값의 평균이 양수로 편향된 문제를 해결하는 함수이지만, 기울기 소멸 문제는 여전히 발생한다.

❯ 그림 4-9 하이퍼볼릭 탄젠트 활성화 함수와 미분 결과



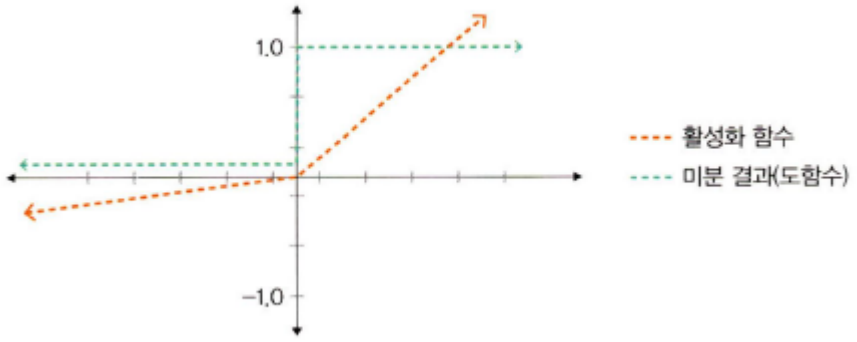
c. **렐루 함수** : 입력이 음수일 때는 0을 출력하고, 양수일 때는 x를 출력한다. 경사 하강법에 영향을 주지 않아 학습 속도가 빠르고, 기울기 소멸 문제가 발생하지 않는다. 렐루 함수는 일반적으로 은닉층에서 사용되며, 하이퍼볼릭 탄젠트 함수 대비 학습 속도가 6배 빠르다. 하지만 음수 값을 입력받으면 항상 0을 출력하기 때문에 학습 능력이 감소한다. 이를 해결하기 위해 리키 렐루 등의 함수를 사용한다.

❯ 그림 4-10 렐루 활성화 함수와 미분 결과



- **리키 렐루 함수** : 입력 값이 음수이면 0이 아닌 0.001처럼 매우 작은 수를 반환한다. 입력 값의 수렴하는 구간이 제거되어 렐루 함수를 사용할 때 생기는 문제를 해결할 수 있다.

❤ 그림 4-11 리키 렐루 활성화 함수와 미분 결과



d. **소프트맥스 함수** : 입력 값이 0에서 1 사이에서 출력되도록 정규화하여 출력 값들의 총 합이 항상 1이 되도록 한다. 보통 딥러닝에서 출력 노드의 활성화 함수로 많이 사용된다.

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

```
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) ## 은닉
    self.relu=torch.nn.ReLu(inplace=True)
    self.out=torch.nn.Linear(n_hidden, n_output) ## 출력층
    self.softmax=torch.nn.Softmax(dim=n_output)
def forward(self, x):
    x=self.hidden(x)
    x=self.relu(x) ## 은닉층을 위한 렐루 활성화 함수
    x=self.out(x)
    x=self.softmax(x) ## 출력층을 위한 소프트맥스 활성화 함수
    return x
```

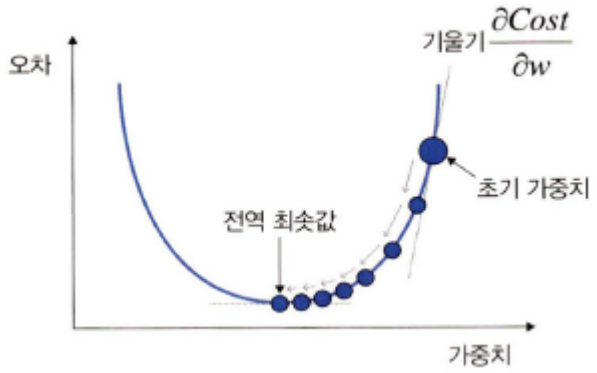
d. **손실 함수 :** 학습을 통해 얻은 데이터의 추정치가 실제 데이터와 얼마나 차이가 나는지 평가하는 지표

- **경사하강법** : 학습률(learning rate)과 손실 함수의 순간 기울기를 이용하여 가중치를 업데이트하는 방법. 즉, 미분의 기울기를 이용하여 오차를 비교하고 최소화하는

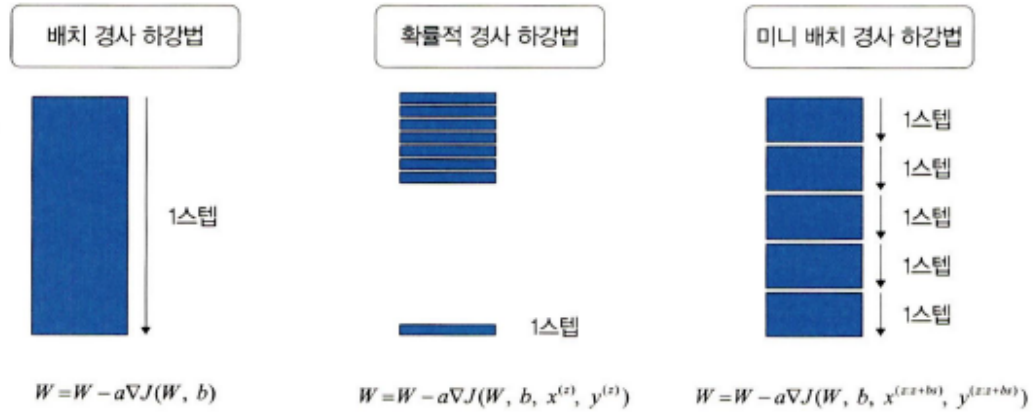
방향으로 이동시키는 방법. 이때 오차를 구하는 방법이 손실 함수이다. 즉, 손실 함수의 비용이 최소가 되는 지점을 찾을 때까지 기울기가 낮은 쪽으로 계속 이동시키는 과정을 반복한다.

- 학습률(learning rate) : 한 번 학습할 때 얼만큼 변화를 주는지에 대한 상수

❮ 그림 4-17 경사 하강법



❮ 그림 4-18 경사 하강법의 유형



1. **배치 경사 하강법(BGD)** : 전체 데이터셋에 대한 오류를 구한 후 기울기를 한 번만 계산하여 모델의 파라미터를 업데이트하는 방법. 즉, 전체 훈련 데이터셋에 대해 가중치를 편미분하는 방법.

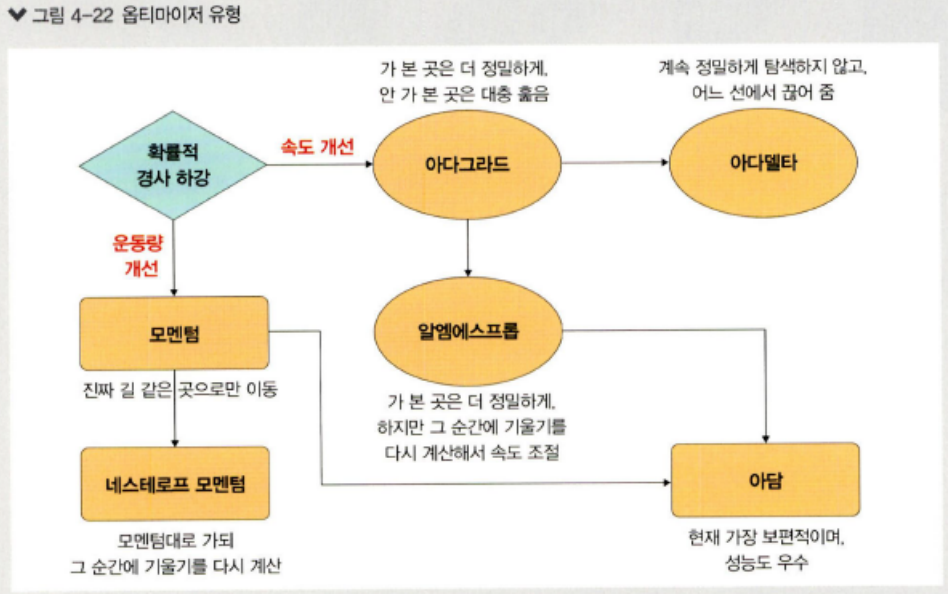
손실 함수의 값을 최소화하기 위해 기울기(▽) 이용

$$W = W - a\nabla J(W, b)$$

(a: 학습률, J: 손실 함수)

한 스텝에 모든 훈련 데이터셋을 사용하므로 학습이 오래 걸린다.

2. **확률적 경사 하강법(SGD)** : 임의로 선택한 데이터에 대해 기울기를 계산하는 방법. 적은 데이터를 사용해서 빠른 계산이 가능하다. 파라미터 변경 폭이 불안정하고, 배치 경사 하강법보다 정확도가 낮게 나오는 경우도 있다.
    - **옵티마이저** : 확률적 경사 하강법의 파라미터 변경 폭이 불안정한 문제를 해결하기 위해 학습 속도와 운동량을 조정하는 역할.



        - 속도 조정 방법
            - 아다그라드(Adagrad) : 변수(가중치)의 업데이트 횟수에 따라 학습률을 조정하는 방법. 많이 변화하지 않는 변수들의 학습률은 크게 하고, 많이 변화하는 변수들의 학습률은 작게 한다. 즉, 많이 변화한 변수는 최적 값에 근접했을 것이라는 가정 하에 작은 크기로 이동하면서 세밀하게 값을 조정하고, 반대로 적게 변화한 변수들은 학습률을 크게 하여 빠르게 오차 값을 줄이고자 한다.

```
optimizer=torch.optim.Adagrad(model.paramet
```

              기울기가 0에 수렴하는 문제가 있어 사용하지 않는다.

            - 아다델타(Adadelta) : 아다그라드에서 기울기 크기의 누적 값이 커짐에 따라 학습이 멈추는 문제를 해결하기 위해 등장한 방법. 아다그라드에서 학습률을 가중치의 변화량 크기를 누적한 값으로 변환한다.

```
optimizer=torch.optim.RMSprop(model.paramet
```

        - 알엠에스프롭(RMSprop) : 아다그라드 개선

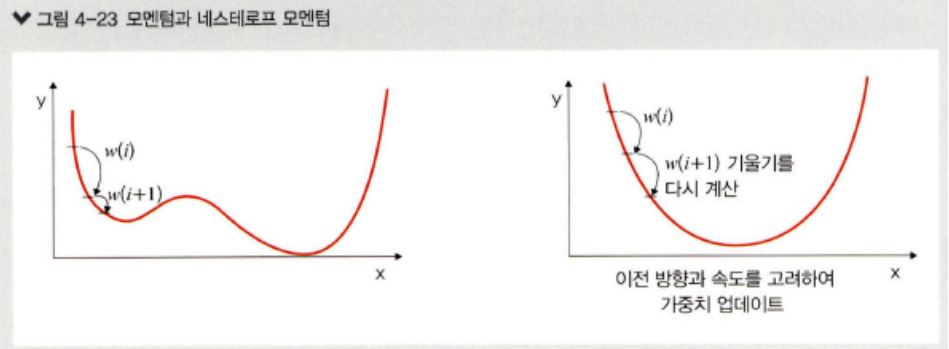
```
optimizer=torch.optim.RMSprop(model.parameters
```

    - 운동량 조절 방법
        - 모멘텀 : 매번 기울기를 구하지만, 가중치를 수정하기 전에 이전 수정 방향을 참고하여 같은 방향으로 일정한 비율만 수정하는 방법. 수정이 양의 방향과 음의 방향으로 반복되는 지그재그 현상이 줄어들고, 이전 이동 값을 고려하여 일정 비율만큼 다음 값을 결정하므로 관성 효과를 얻을 수 있다.

```
optimizer=torch.optim.SGD(model.parameters(),
```

        - 네스테로프 모멘텀(NAG) : 모멘텀 값과 기울기 값이 더해져 실제 값을 만드는 기존 모멘텀과 달리 모멘텀 값이 적용된 지점에서 기울기 값을 계산한다. 이동 속도가 빠르고, 멈추어야 할 적절한 시점에서 제동을 거는 데 용이하다.

```
optimizer=torch.optim.SGD(model.parameters(),
```



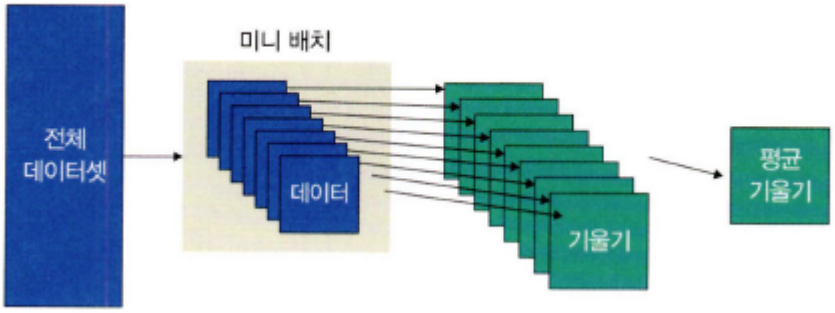
    - 속도와 운동량에 대한 혼용 방법 : 아담(Adam)

모멘텀과 알엠에스프롭의 장점을 결합한 경사 하강법.

```
optimizer=torch.optim.Adam(model.parameters(), lr
```

3. **미니 배치 경사 하강법** : 전체 데이터셋을 미니 배치 여러 개로 나누고, 미니 배치 한 개마다 기울기를 구한 후 그것의 평균 기울기를 이용하여 모델을 업데이트해서 학습하는 방법

❯ 그림 4-20 미니 배치 경사 하강법



전체 데이터를 계산하는 것보다 빠르고, 확률적 경사 하강법보다 안정적이기 때문에 실제로 많이 사용한다.

```
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, ## 데이터셋
    batch_size=2, ## 미니 배치 크기로 2의 제곱수를 사용한다
    shuffle=True, ## 데이터를 불러올 때마다 랜덤으로 섞어서
)
```

a. **평균 제곱 오차(MSE)** : 실제 값과 예측 값의 차이를 제곱하여 평균을 낸 것. 실제 값과 예측 값의 차이가 클수록 평균 제곱 오차의 값도 커진다. 이 값이 작을수록 예측력이 좋다. 회귀에서 손실 함수로 주로 사용된다.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

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

```
import torch

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

b. **크로스 엔트로피 오차(CEE)** : 분류 문제에서 원-핫 인코딩했을 때 사용할 수 있는 오차 계산법

일반적으로 분류 문제에서는 데이터의 출력을 0과 1로 구분하기 위해 `시그모이드 함수`를 사용한다. 하지만 `시그모이드 함수`에 포함된 자연 상수 e 때문에 평균 제곱 오차를 적용하면 매끄럽지 못한 그래프가 출력된다.

→ 손실 함수로 크로스 엔트로피 손실 함수 사용. 이 손실 함수를 적용할 경우 경사 하강법 과정에서 학습이 `지역 최소점`에서 멈출 수 있다. 이것을 방지하고자 자연 상수 e에 반대되는 자연 로그를 모델의 출력 값에 취한다.

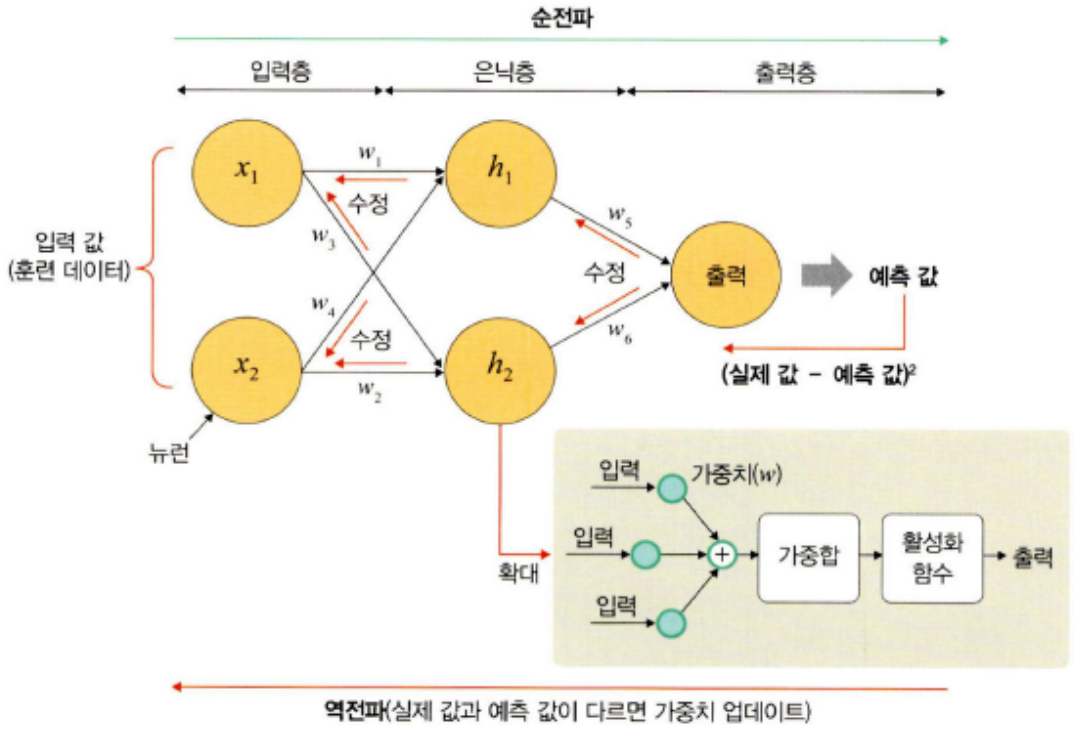
$$CrossEntropy = -\sum_{i=1}^{n} y_i \log \hat{y}_i$$

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

```
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()
```

4.2.2 딥러닝 학습

❤ 그림 4-12 순전파와 역전파



1단계. 순전파 : 네트워크에 훈련 데이터가 들어올 때 발생하며, 데이터를 기반으로 예측 값을 계산하기 위해 전체 신경망을 교차해 지나간다. 즉, 모든 뉴런이 이전 층의 뉴런에서 수신한 정보에 변환(가중합 및 활성화 함수)을 적용하여 다음 층(은닉층)의 뉴런으로 전송하는 방식이다. 네트워크를 통해 입력 데이터를 전달하며, 데이터가 모든 층을 통과하고 모든 뉴런이 계산을 완료하면 그 예측 값은 최종 층(출력층)에 도달하게 된다.

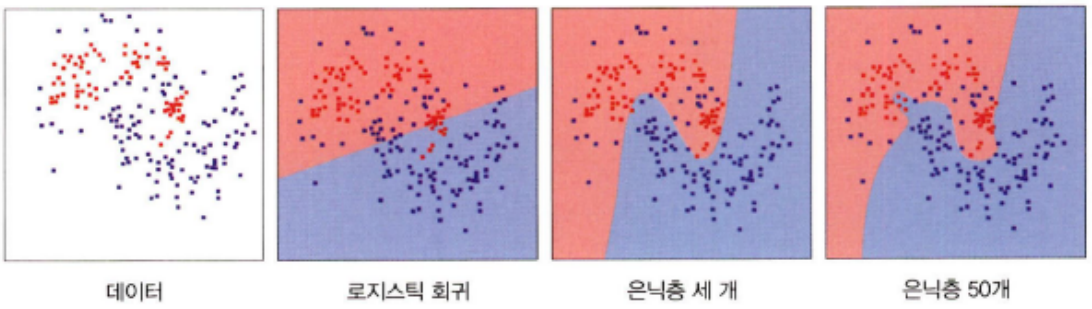
2단계. 손실 함수 : 네트워크의 예측 값과 실제 값의 차이(손실, 오차)를 추정한다. 이때 손실 함수 비용은 0이 이상적이다. 따라서 손실 함수 비용이 0에 가깝도록 하기 위해 모델이 훈련을 반복하면서 가중치를 조절한다.

3단계. 역전파 : 손실(오차)이 계산되면 그 정보는 역으로 전파(출력층 → 은닉층 → 입력층)된다. 출력층에서 시작된 손실 비용은 은닉층의 모든 뉴런으로 전파되지만, 은닉층의 뉴런은 각 뉴런이 원래 출력에 기여한 상대적 기여도에 따라(즉, 가중치에 따라) 값이 달라진다. 다시 말하면 예측 값과 실제 값의 차이를 각 뉴런의 가중치로 미분한 후 기존 가중치 값에서 뺀다. 이 과정을 출력층 → 은닉층 → 입력층 순서로 모든 뉴런에 대해 진행하여 계산된 각 뉴런 결과를 다시 순전파의 가중치 값으로 사용한다.

4.2.3 딥러닝의 문제점과 해결 방안

딥러닝의 핵심은 활성화 함수가 적용된 여러 은닉층을 결합하여 비선형 영역을 표현하는 것이다. 활성화 함수가 적용된 은닉층 개수가 많을수록 데이터 분류가 잘 된다.

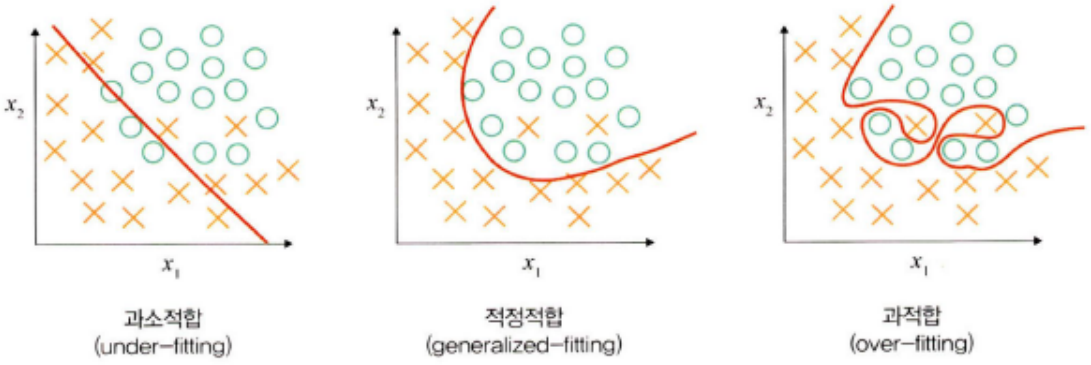
❯ 그림 4-13 은닉층이 분류에 미치는 영향



하지만 은닉층의 개수가 많아질수록 문제점 또한 늘어난다.

1. **과적합 문제** : 훈련 데이터를 과하게 학습해서 발생한다. 훈련 데이터를 과하게 학습하면 예측 값과 실제 값의 차이인 오차가 감소하지만, 검증 데이터에 대해서는 오차가 증가할 수 있다. 이러한 관점에서 과적합은 훈련 데이터에 대해 과하게 학습하여 실제 데이터에 대한 오차가 증가하는 현상을 의미한다.

❯ 그림 4-14 과적합



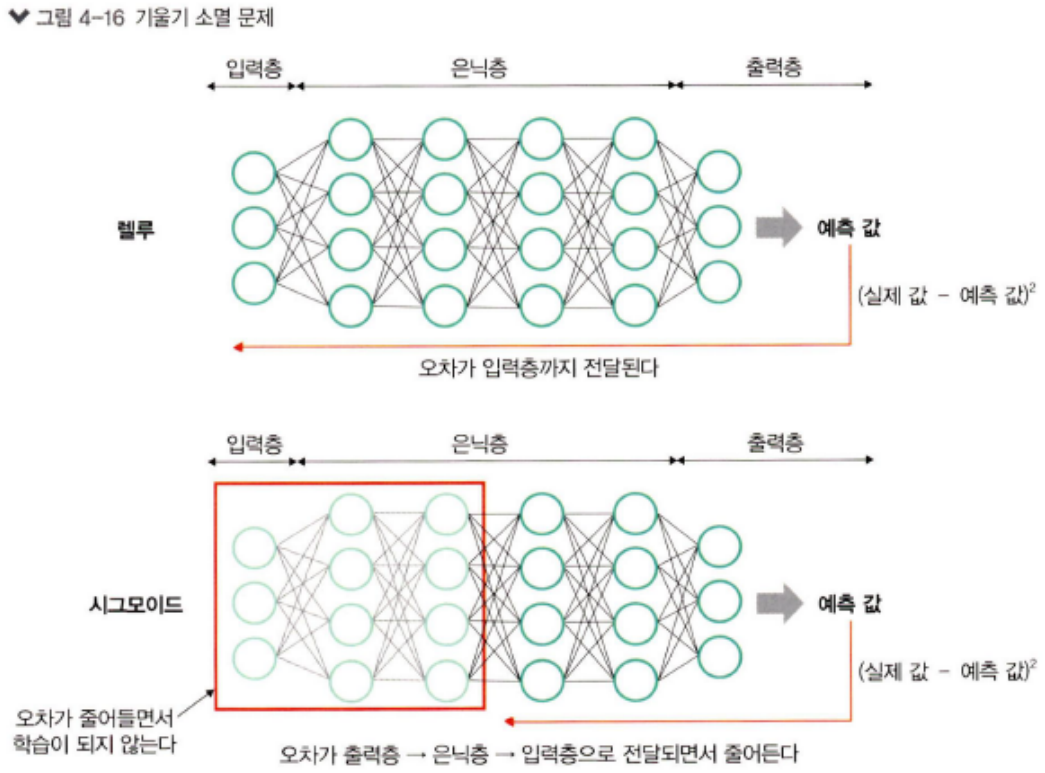
→ **드롭아웃(dropout)** : 신경망 모델이 과적합되는 것을 피하기 위한 방법으로, 학습 과정 중 임의로 일부 노드들을 학습에서 제외시킨다.

```
class DropoutModel(torch.nn.Module):
    def __init__(self):
        super(DropoutModel, self).__init__()
        self.layer1=torch.nn.Linear(784, 1200)
        self.dropout1=torch.nn.Dropout(0.5) ## 50%의 노드를
        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)
```

2. 기울기 소멸 문제 : 은닉층이 많은 신경망에서 발생. 출력층에서 은닉층으로 전달되는 오차가 크게 줄어들어 학습이 되지 않는 현상. 즉, 기울기가 소멸되기 때문에 학습되는 양이 0에 가까워져 학습이 더디게 진행되다 오차를 더 줄이지 못하고 그 상태로 수렴하는 현상.



→ 렐루 활성화 함수 사용

3. **성능이 나빠지는 문제** : 경사 하강법 시 발생하는 문제.

→ 미니 배치 경사 하강법

4.2.4 딥러닝을 사용할 때 이점

1. 특성 추출 : 컴퓨터가 입력받은 데이터를 분석하여 일정한 패턴이나 규칙을 찾아내려면 사람이 인지하는 데이터를 컴퓨터가 인지할 수 있는 데이터로 변환해야 한다. 이때 데이터별로 어떤 특징을 갖고 있는지 찾아내고, 그것을 토대로 벡터로 변환하는 작업.

2. 빅데이터의 효율적 활용 : 딥러닝 학습을 이용한 특성 추출은 데이터 사례가 많을수록 성능이 향상된다.

[딥러닝 파이토치 교과서] 4장 딥러닝 시작(4.3)

작성일시	@2024년 9월 29일 오후 2:39
분야	DL
주제	기초
type	필사
날짜	@2024년 9월 29일

4.3 딥러닝 알고리즘

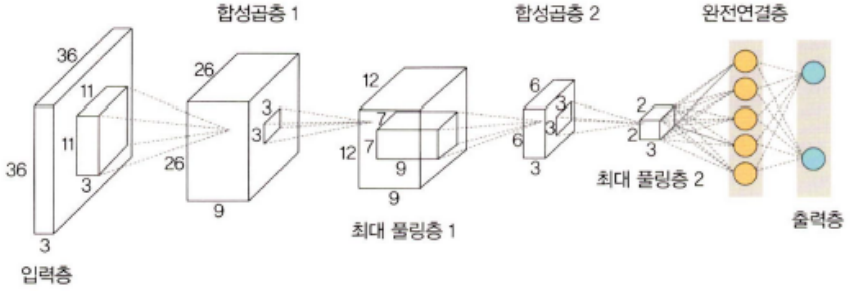
4.3.1 심층 신경망

→ 심층 신경망(Deep Neural Network, DNN, 딥러닝) : 입력층과 출력층 사이에 은닉층이 여러 개 있는 신경망

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 객체를 탐색하거나 객체 위치를 찾아내는 데 유용한 신경망이다.

그림 4-25 합성곱 신경망

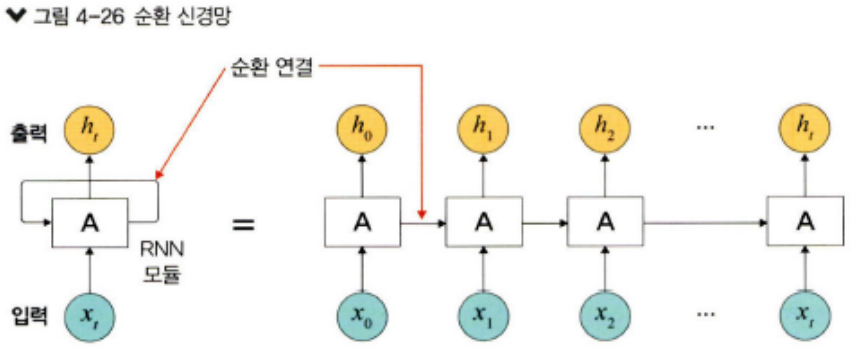


- 종류 : LeNet-5, AlexNet, VGG, GoogLeNet, ResNet
- 차별성
  - 각 층의 입출력 형상 유지
  - 이미지 공간 정보를 유지하면서 인접 이미지와 차이가 있는 특징을 효과적으로 인식한다.
  - 복수 필터로 이미지의 특징을 추출하고 학습한다.
  - 추출한 이미지의 특징을 모으고 강화하는 풀링층이 있다.
  - 필터를 공유 파라미터로 사용하기 때문에 일반 인공 신경망과 비교하여 학습 파라미터가 매우 적다.

4.3.3 순환 신경망

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

- 순환 : 자기 자신을 참조한다. 즉, 현재 결과가 이전 결과와 연관이 있다.



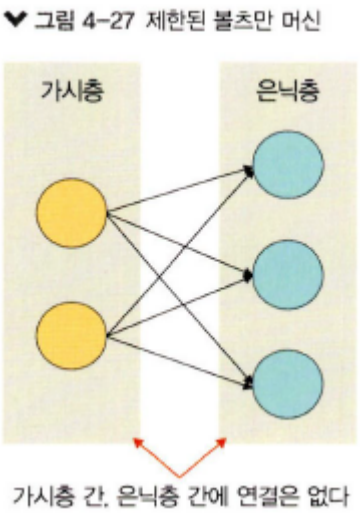
- 특징
  - 시간성 데이터가 많다. 동적이고, 길이가 가변적이다.
  - 매우 긴 데이터를 처리하는 연구가 활발히 진행되고 있다.

순환 신경망은 기울기 소멸 문제로 학습이 제대로 되지 않는 문제가 있다. 이를 해결하고자 메모리 개념을 도입한 LSTM이 순환 신경망에서 많이 사용되고 있다. 자연어 처리 분야와 연관이 있다.

4.3.4 제한된 볼츠만 머신

볼츠만 머신 : 가시층, 은닉층으로 구성된 모델

제한된 볼츠만 머신 : 가시층이 은닉층과만 연결되는 모델



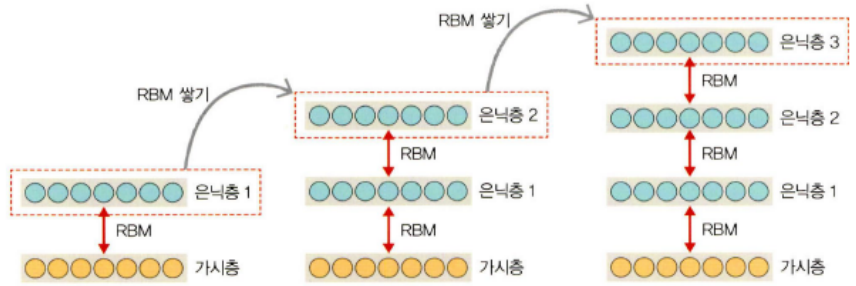
- 특징
  - 차원 감소, 분류, 선형 회귀 분석, 협업 필터링, 특성 값 학습, 주제 모델링에 사용한다.
  - 기울기 소멸 문제를 해결하기 위해 사전 학습 용도로 활용할 수 있다.
  - 심층 신뢰 신경망(DBN)의 요소로 활용된다.

4.3.5 심층 신뢰 신경망

입력층과 은닉층으로 구성된 제한된 볼츠만 머신을 블록처럼 여러 층으로 쌓은 형태로 연결된 신경망. 즉, 사전 훈련된 제한된 볼츠만 머신을 층층이 쌓아 올린 구조이다. 레이블이 없는 데이터에 대한 비지도 학습이 가능하다. 부분적인 이미지에서 전체를 연상하는 일반화와 추상화 과정을 구성할 때 사용할 수 있다.

- 학습 절차
  1. 가시층과 은닉층 1에 제한된 볼츠만 머신을 사전 훈련한다.
  2. 첫 번째 층 입력 데이터와 파라미터를 고정하여 두 번째 층 제한된 볼츠만 머신을 사전 훈련한다.
  3. 원하는 층 개수만큼 제한된 볼츠만 머신을 쌓아 올려 전체 DBN을 완성한다.

❯ 그림 4-28 심층 신뢰 신경망



- 특징
  - 순차적으로 심층 신뢰 신경망을 학습시켜 가면서 계층적 구조를 생성한다.
  - 비지도 학습으로 학습한다.
  - 위로 올라갈수록 추상적 특성을 추출한다.
  - 학습된 가중치를 다층 퍼셉트론의 가중치 초깃값으로 사용한다.