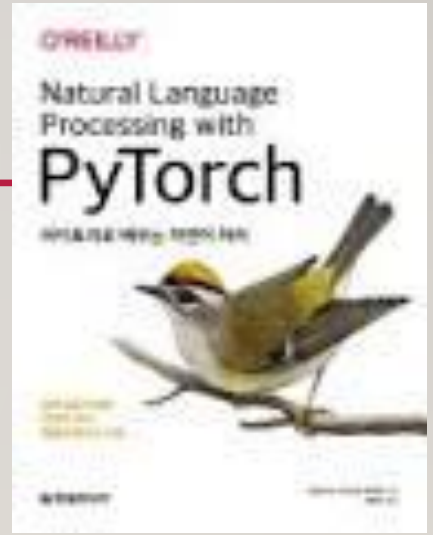
파이토치로 배우는 자연어 처리

3장. 신경망의 기본 구성 요소

4장. 자연어 처리를 위한 피드포워드 신경망



AI융합학부 20193124 고경빈

목차

퍼셉트론(PERCEPTRON)

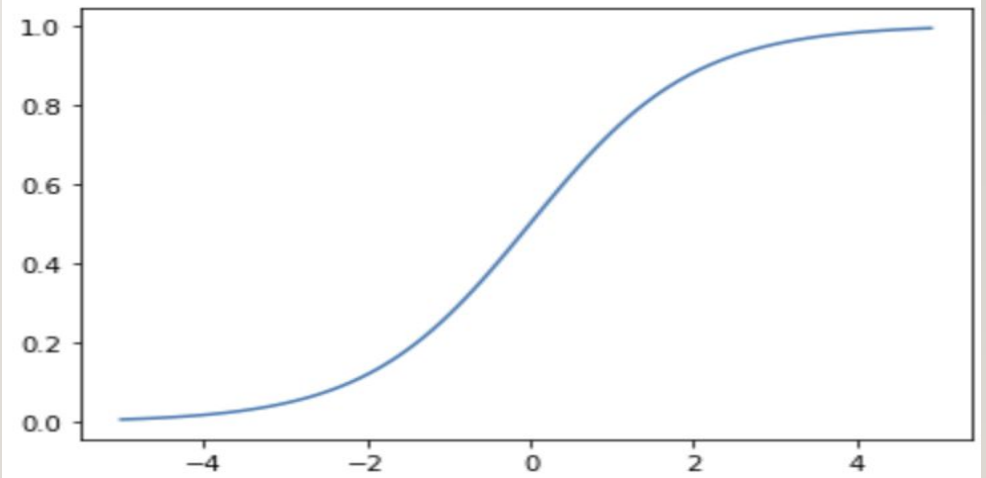
```python
import torch
import torch.nn as nn

class Perceptron(nn.Module):
    def __init__(self, input_dim):
        super(Perceptron, self).__init__()
        self.fc1 = nn.Linear(input_dim, 1)

    def forward(self, x_in):
        return torch.sigmoid(self.fc1(x_in)).squeeze()
```

- 가장 간단한 신경망으로 생물학적 뉴런을 본떠 만듦
- 입력과 출력이 있고 신호는 입력에서 출력으로 흐름
- 입력(x), 출력(y), 가중치(w), 절편(b), 활성화 함수(f)로 구성
- 가중치와 절편은 데이터에서 학습됨
- y=f(wx+b)
- 선형 함수와 비선형 함수의 조합

시그모이드

- 임의의 실숫값을 받아 0과 1사이의 범위로 압축
- $f(x) = \frac{1}{1+e^{-x}}$
- 입력 범위 대부분에서 빠르게 포화됨
  - 그레디언트 소실/폭주 문제 발생
- 출력층에서 출력을 확률로 압축하는 데 사용

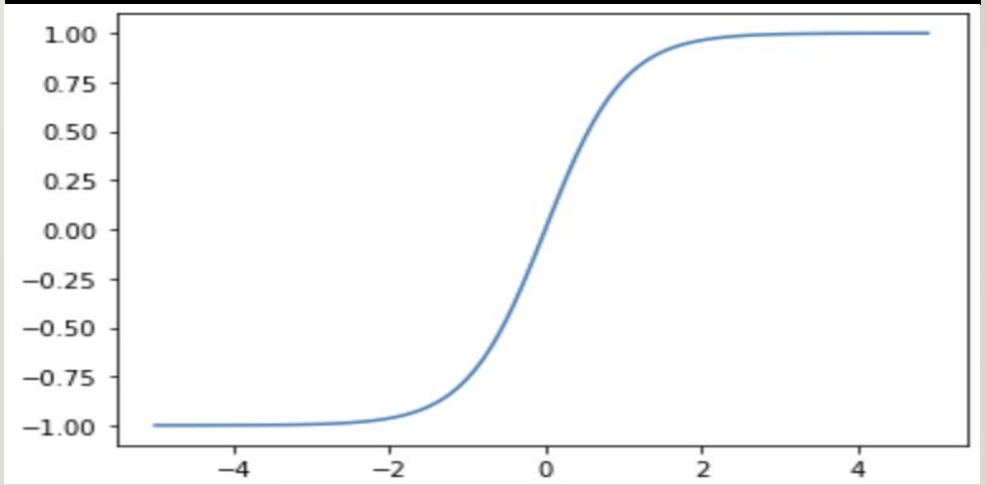
```
x = torch.arange(-5., 5., 0.1)
y = torch.sigmoid(x)
plt.plot(x.numpy(), y.detach().numpy())
plt.show()
```



하이퍼볼릭 탄젠트

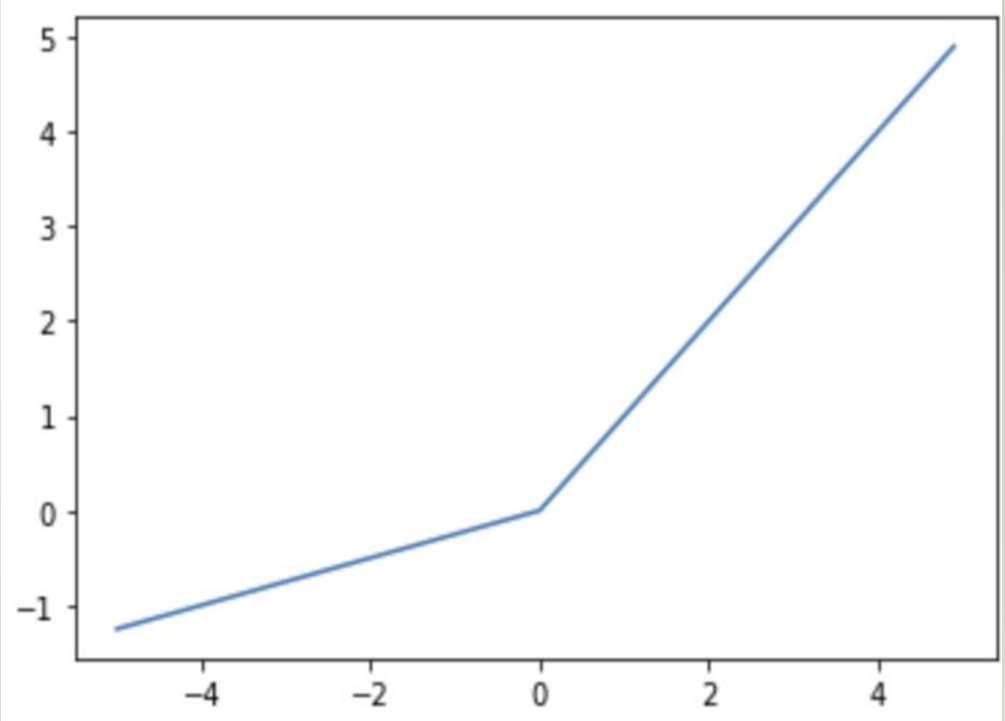
- 시그모이드 함수의 변종
- $f(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- 압축 함수이지만, $(-\infty, \infty)$ 범위의 실숫값을 $[-1, 1]$로 바꾸는 점이 다름

```
x = torch.arange(-5., 5., 0.1)
y = torch.tanh(x)

plt.plot(x.numpy(), y.detach().numpy())
plt.show()
```



렐루

- 가장 중요한 활성화 함수
- $f(x) = \max(0, x)$
- 음숫값을 0으로 자름
- 음수를 제거해 그레디언트 소실 문제에 도움이 됨
- 신경망의 특정 출력이 0이 되면 다시 돌아오지 못한다는 문제(죽은 렐루)
  - 변종 렐루(Leaky ReLU, PReLU)



소프트맥스

- 신경망 유닛의 출력을 0과 1 사이로 압축
- 모든 출력의 합으로 각 출력을 나누어 k개 클래스에 대한 이산 확률 분포를 만듦
- $softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} x_j}$
- 분류 작업의 출력을 해석하는데 유용
- 확률 기반의 목적 함수인 범주형 크로스 엔트로피와 함께 사용

```
softmax = nn.Softmax(dim=1)
x_input = torch.randn(1, 3)
y_output = softmax(x_input)
print(x_input)
print(y_output)
print(torch.sum(y_output, dim=1))
```

```
tensor([[-2.0260, -2.0655, -1.2054]])
tensor([[0.2362, 0.2271, 0.5367]])
tensor([1.])
```

평균 제곱 오차(MSE)

- 신경망의 출력과 타깃의 연속적인 회귀 문제에서 널리 사용되는 손실 함수
- $L_{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2$
- 예측과 타깃값의 차이를 제곱하여 평균한 값

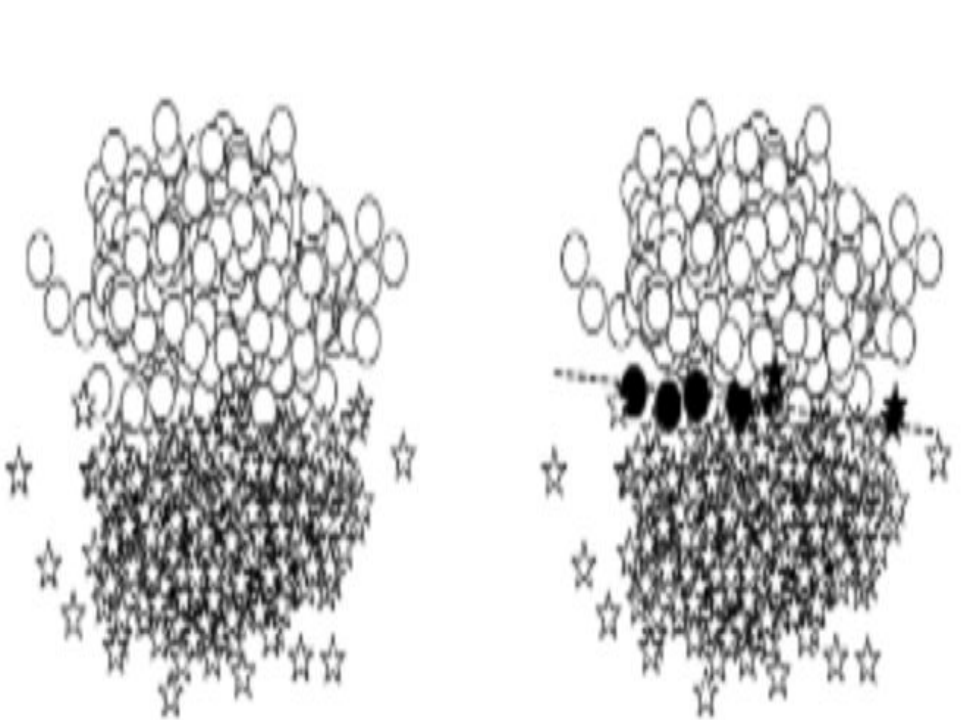
이진 크로스 엔트로피 손실

- 클래스 두 개를 구별하는 작업에 효율적

범주형 크로스 엔트로피 손실

- 출력을 클래스 소속 확률에 대한 예측으로 이해할 수 있는 다중 분류 문제에 사용
- 신경망의 출력은 원소 n 개로 구성된 벡터이며 다항 분포에 대한 신경망의 예측을 나타냄
- $L_{cross_{entropy}}(y, \hat{y}) = \sum_i y_i \log(\widehat{y_i})$
- 신경망 출력과 손실 함수 간의 관계를 결정하는 정보
  - 수의 범위 제한
  - 소프트맥스 함수에 사용한 지수 함수의 입력이 음수이면 그 결과는 작은 수가 되고 양수이면 큰 수가 됨
  - 신경망의 출력은 소프트맥스 함수를 적용하기 직전의 벡터라고 가정
  - 로그 함수는 지수 함수의 역함수이고 log(exp(x))는 x와 같음

지도 학습 훈련



- 퍼셉트론은 어떤 크기의 입력도 다룰 수 있음
- 퍼셉트론의 활성화 함수가 시그모이드이므로 퍼셉트론의 출력은 $P(y=1|x)$
- 예측 확률 $P(y=1|x) > k$이면 예측 클래스는 1이고, 아니면 0
- 모델이 이진 출력을 만드므로 이진 크로스 엔트로피 손실 함수를 사용
- 옵티마이저
  - 모델이 예측을 만들고 손실 함수가 예측과 타깃의 오차를 측정하면 이 오차 신호를 사용해 모델의 가중치를 업데이트함

그레디언트 업데이트

- 모델 객체 안에 저장된 그레디언트와 같은 부가 정보를 zero_grad() 함수로 초기화
- 모델이 입력 데이터에 대한 출력을 계산
- backward() 메서드를 사용해 계산 그래프를 거슬러 손실을 반복해서 전파하고 각 파라미터에 대한 그레디언트를 계산
- 옵티마이저는 step() 함수로 파라미터에 그레디언트를 업데이트 하는 방법을 지시

부가적인 훈련 개념(I)

- 평가 지표
  - 모델이 훈련에 사용하지 않은 데이터를 사용해 성능을 측정(ex. 정확도)
- 데이터 분할
  - 훈련 데이터의 샘플 뿐만 아니라 본 적 없는 분포의 샘플에서도 오차를 줄일 때 일반화가 잘된 모델이라고 함
  - 손실을 낮출수록 진짜 데이터 분포에 없는 특징에 적응하여 과대적합이 될 수 있음
  - 데이터셋을 훈련/검증/테스트(7:1.5:1.5)로 나누거나 K-겹 교차 검증을 사용하는 것이 표준
  - 테스트 데이터를 많이 사용하면 오버피팅 문제가 발생
  - k-겹 교차 검증은 계산 비용이 많이 들지만 작은 데이터셋에 유용

부가적인 훈련 개념(2)

- 훈련 중지 시점 파악하기
  - 고정된 에포크 횟수만큼 훈련하는 방법
  - 조기 종료: 성능이 계속 좋아지지 않으면 훈련을 종료
- 최적의 하이퍼 파라미터 찾기
  - 모델의 파라미터 개수와 값에 영향을 미치는 모든 모델 설정
  - 옵션의 선택이 모델의 수렴과 성능에 큰 영향을 미칠 수 있으므로 여러 선택 옵션을 체계적으로 테스트해야 함

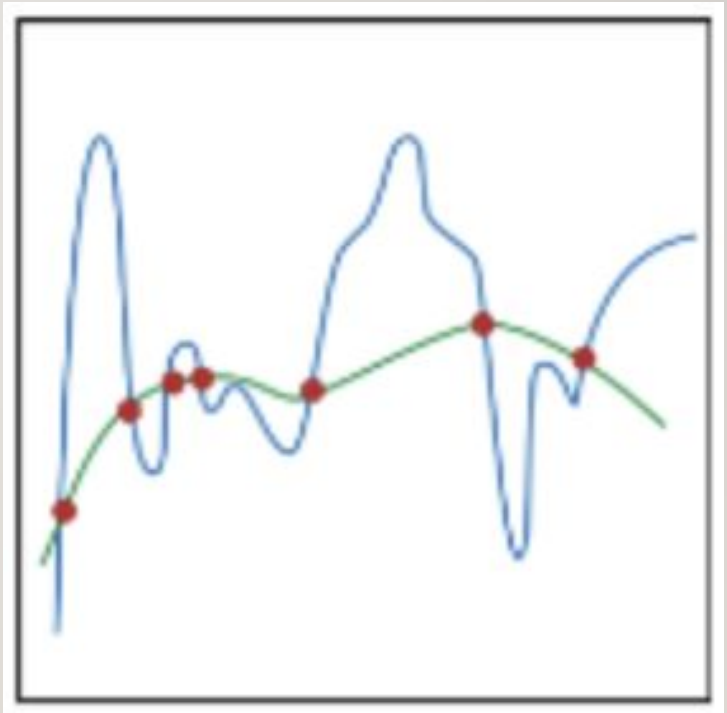
규제

- L2 규제
  - 부드럽게 만드는 제약

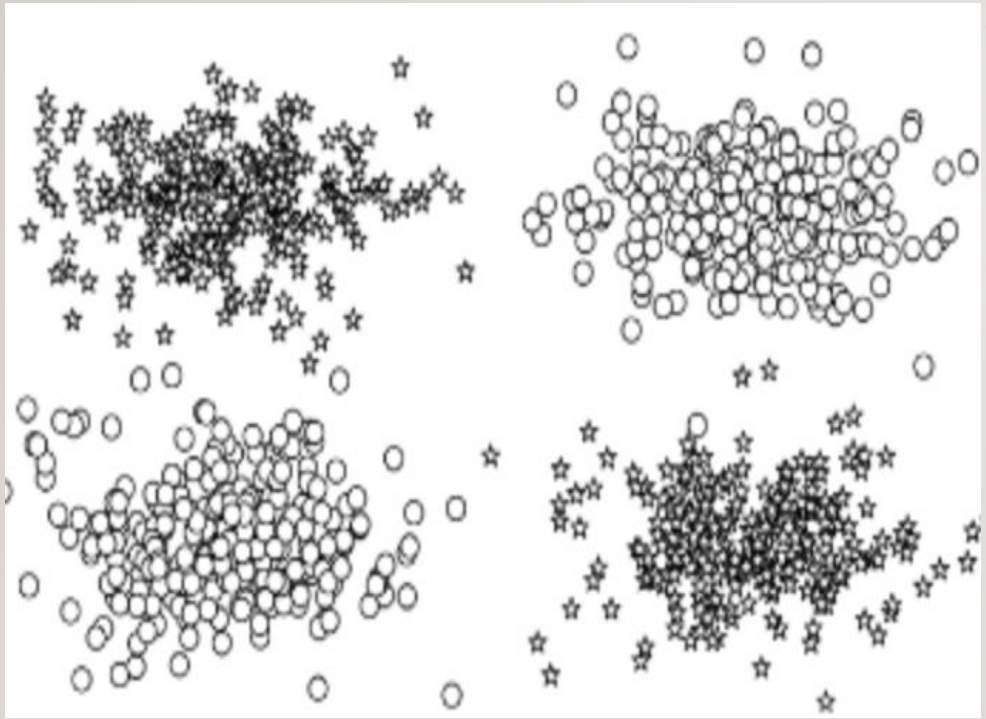
$$L2 : \|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{m} w_j^2$$

- L1 규제
  - 희소한 솔루션을 만드는 데 사용

$$L1 : \|\boldsymbol{w}\|_1 = \sum_{j=1}^{m} |w_j|$$



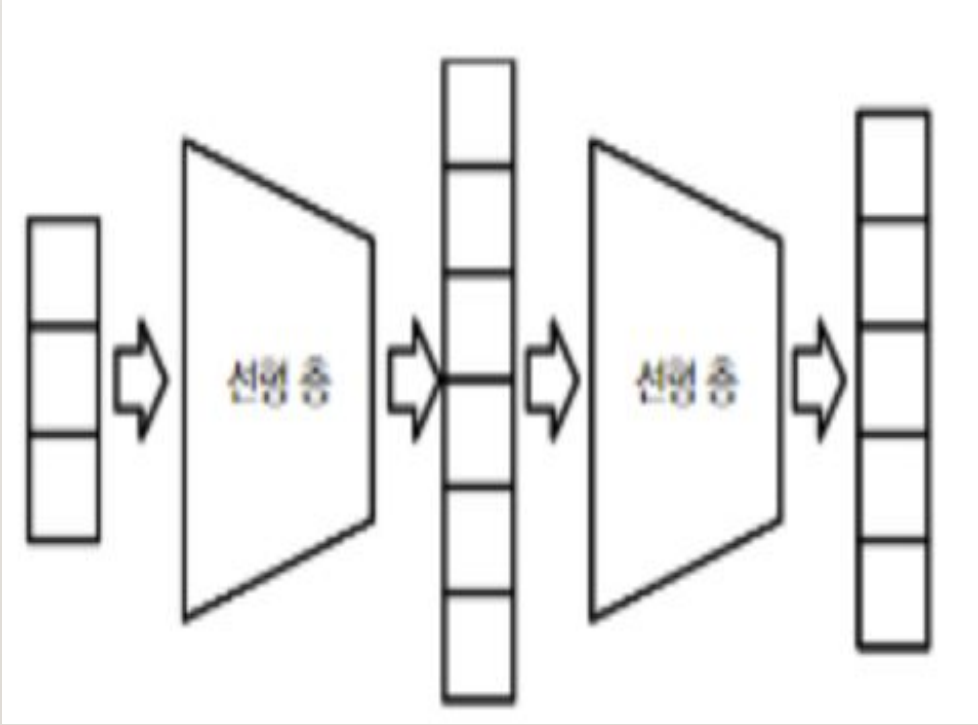
피드포워드 신경망



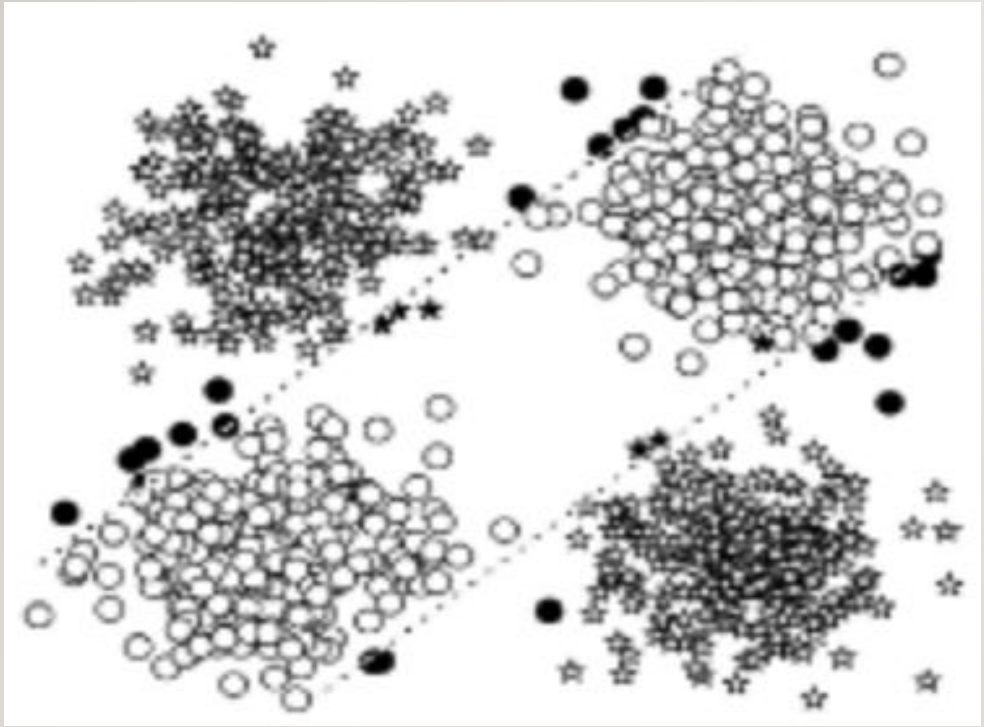
- 피드포워드 신경망
  - 다층 퍼셉트론(MLP)
    - 많은 퍼셉트론이 있는 층을 여러 개 쌓아 올린 구조
  - 합성곱 신경망(CNN)
    - 입력에 있는 국부 패턴을 학습할 수 있고, 컴퓨터 비전에 적합하고 순차 데이터에서 부분 구조를 감지하는 데 이상적

다층 퍼셉트론

- 층의 출력은 출력값 하나가 아닌 벡터
- 층 사이에 비선형성을 추가
- MLP의 강력한 성능은 두 번째 Linear 층을 추가하여 모델이 중간 표현을 학습할 수 있는 데서 비롯됨



XOR

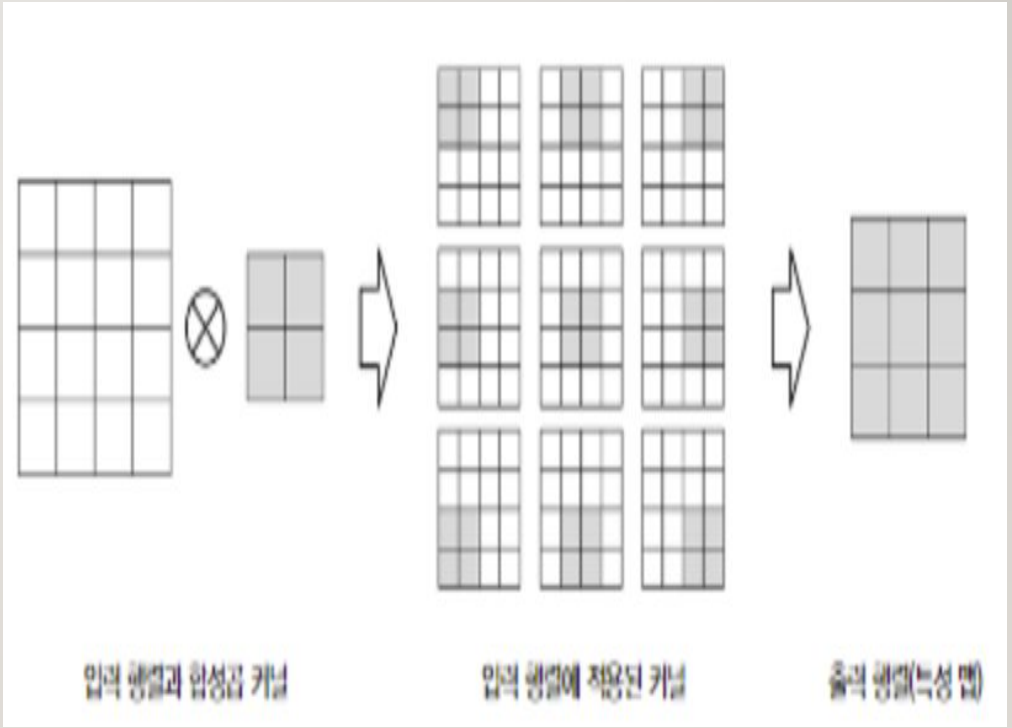


- 퍼셉트론의 약점이 잘 보완됨
- 중간 표현이 공간을 변형해 초평명 하나가 두 곳에 나타나도록 만들어서 결정 경계가 이렇게 보임

```
class MultilayerPerceptron(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(MultilayerPerceptron, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x_in, apply_softmax=False):
        intermediate = F.relu(self.fc1(x_in))
        output = self.fc2(intermediate)

        if apply_softmax:
            output = F.softmax(output, dim=1)
        return output
```

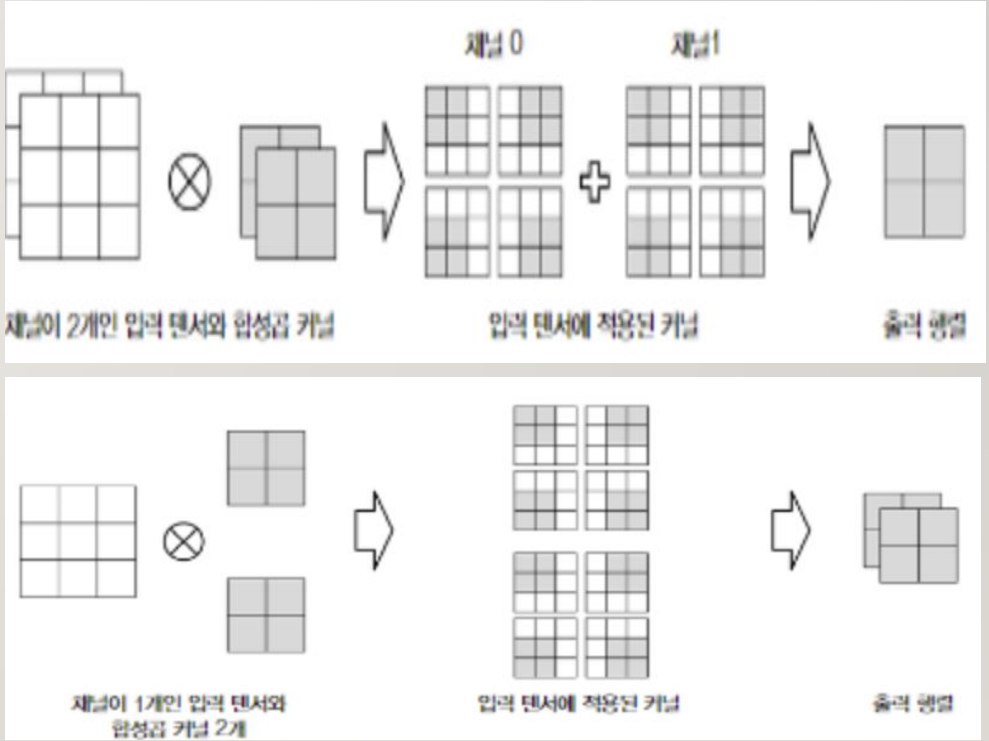
CNN 하이퍼 파라미터

- 하이퍼 파라미터를 지정해 CNN의 동작을 제어하고 경사 하강법으로 주어진 데이터셋에서 최적의 파라미터를 찾음
- 합성곱 연산의 차원
  - 1차원: 시계열에 잘 맞음, 시퀀스를 차원에 따라 패턴을 학습할 수 있음
  - 2차원: 데이터의 두 방향을 따라 시공간 패턴을 감지, 이미지 처리 분야에서 인기
  - 3차원: 데이터의 세 방향을 따라 패턴을 감지



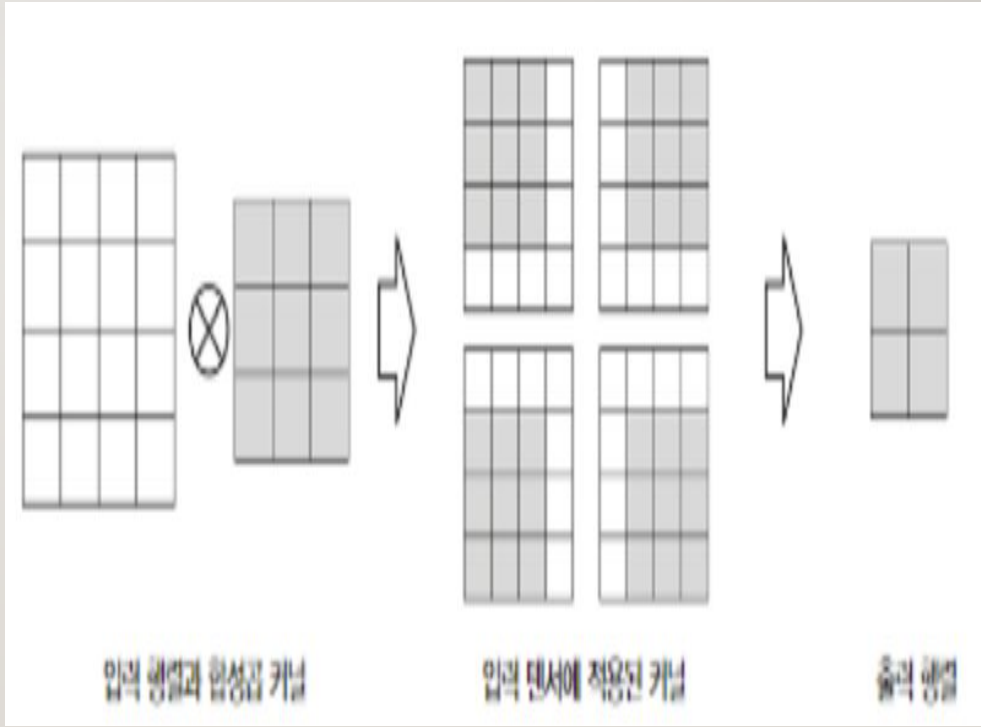
채널

- 입력의 각 포인트에 있는 특성 차원을 의미
- 텍스트 문서의 픽셀이 단어라면 채널 개수는 어휘 사전의 크기
- 문자에 대한 합성곱을 수행한다면 채널 개수는 문자 집합의 크기



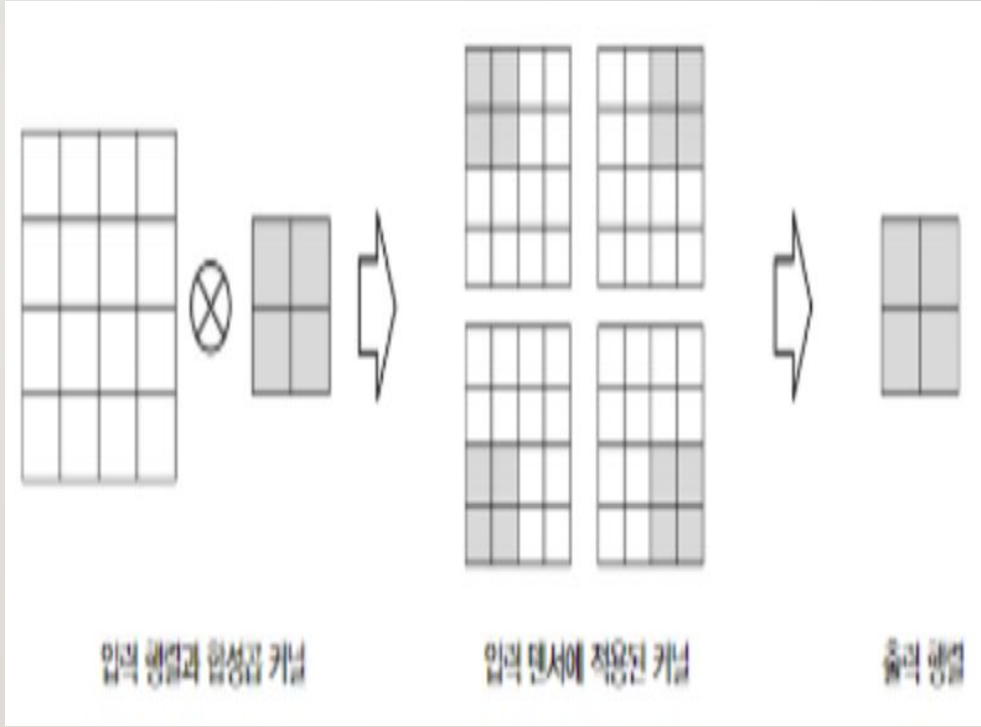
커널 크기

- 커널 행렬의 너비를 의미
- 합성곱마다 얻어지는 정보의 양은 커널 크기로 조절
- 커널이 행렬에 적용될 때 더 많은 국부적인 정보가 사용되지만 출력 크기는 작아짐
- 커널 크기가 작을수록 작고 자주 등장하는 패턴을 감지
- 커널 크기가 커지면 큰 패턴을 감지



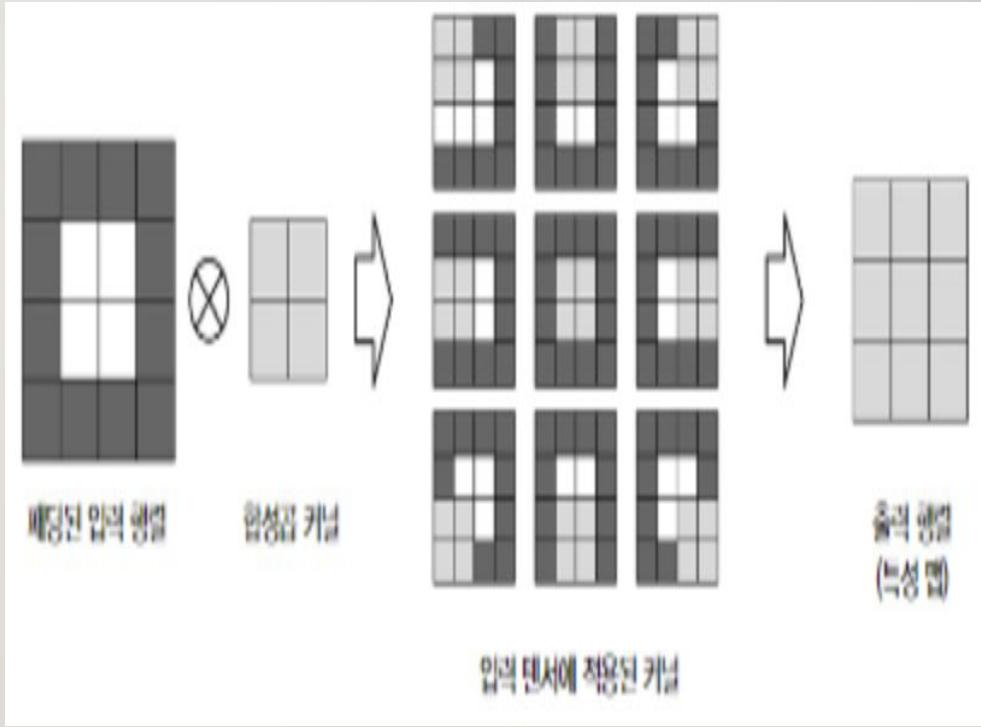
스트라이드

- 합성곱 간의 스텝 크기를 제어
- 스트라이드가 커널 크기와 같으면 커널 연산이 겹치지 않음
- 스트라이드를 높여 출력 텐서의 크기를 의도적으로 줄여서 정보를 요약할 수 있음



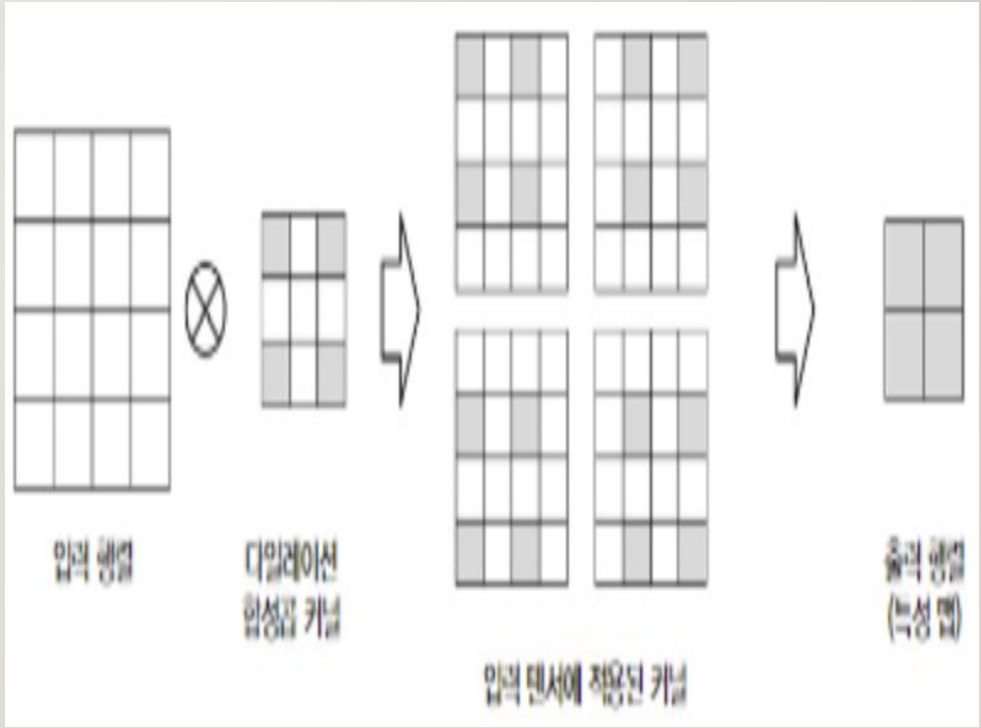
패딩

- 스트라이드와 커널 사이즈는 계산된 특성값의 범위를 제어하지만, 특성 맵의 전체 크기를 의도치 않게 줄이는 현상 발생
  - 입력 텐서의 길이, 높이, 깊이 차원의 앞뒤에 0을 추가하여 인공적으로 늘림



다일레이션

- 합성곱 커널이 입력 행렬에 적용되는 방식을 제어
- 다일레이션을 1에서 2로 늘리면 입력 행렬에 적용될 때 커널의 원소 사이에 공간이 생김
- 파라미터 개수를 늘리지 않고 넓은 입력 공간을 요약하는 데 유용
- 층을 쌓을 때 매우 유용
  - 연속된 다일레이션 합성곱은 수용장의 크기를 기하급수적으로 늘려줌



출력 텐서의 크기를 줄이는 방법(I)

- 합성곱 층을 더 만들어 차례대로 적용하는 방법
- 합성곱마다 채널 차원의 크기는 증가
  - 텐서가 특성 벡터가 되는 최종 단계는 불필요한 size=1 차원을 제거하는 것

```
conv2 = nn.Conv1d(in_channels=16, out_channels=32, kernel_size=3)
conv3 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3)

intermediate2 = conv2(intermediate1)
intermediate3 = conv3(intermediate2)

print(intermediate2.size())
print(intermediate3.size())
```

```
torch.Size([2, 32, 3])
torch.Size([2, 64, 1])
```

출력 텐서의 크기를 줄이는 방법(2,3)

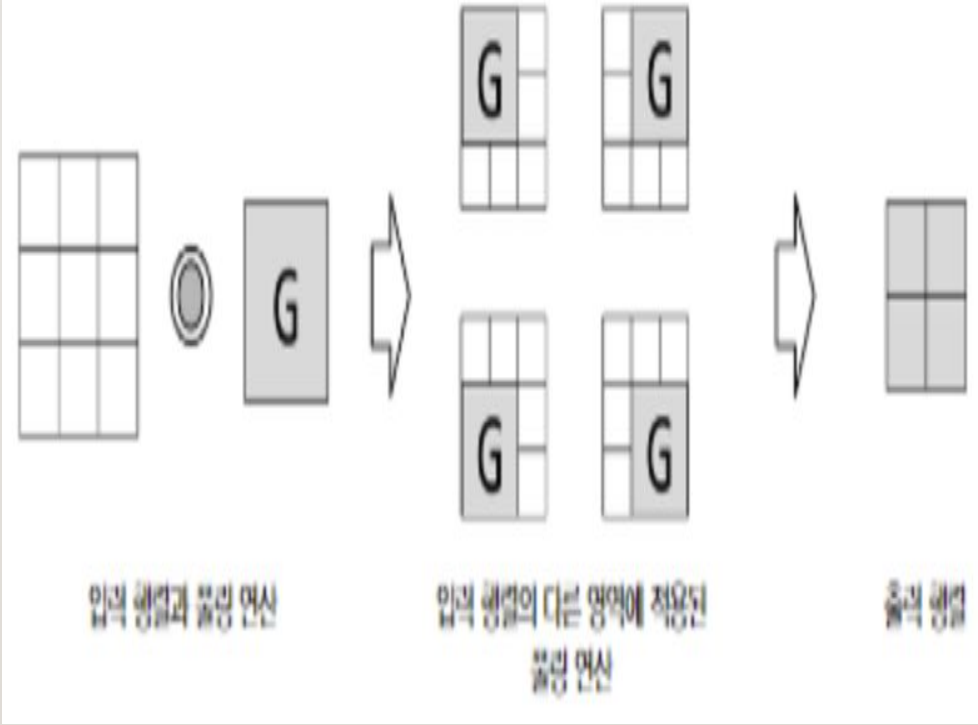
- 남은 값을 특성 벡터로 펼치는 방법
  - 모든 정보를 유지하지만 필요 이상으로 큰 특성 벡터를 만들기도 함
- 특성 맵 차원을 따라 평균을 계산하는 방법
  - 평균은 특성 맵 차원의 크기에 상관없지만 일부 정보를 잃을 수 있음

```
# 특성 벡터를 줄이는 방법 2
print(intermediate1.view(batch_size, -1).size())

# 특성 벡터를 줄이는 방법 2
print(torch.mean(intermediate1, dim=2).size())
# print(torch.max(intermediate1, dim=2).size())
# print(torch.sum(intermediate1, dim=2).size())
```

```
torch.Size([2, 80])
torch.Size([2, 16])
```

풀링

- 합성곱 연산의 중첩되는 특징 때문에 많은 특성이 중복될 수 있음
- 고차원이고 중복 가능성이 높은 특성 맵을 저차원으로 요약하는 방법
  - 합(sum),평균(average), 최대(max) 풀링이 있음
  - 통계적으로 크기가 큰 특성 맵을 강하고 작은 특성맵으로 개선할 수 있음

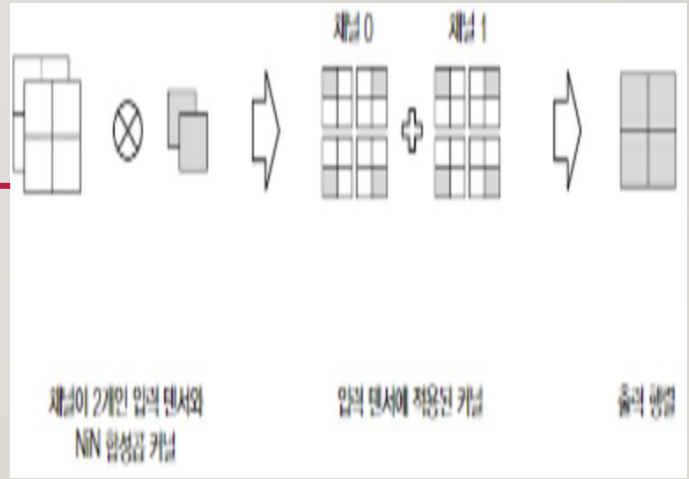


배치 정규화

- CNN의 출력에 적용되어 활성화 값이 평균이 0이고 단위 분산이 되도록 만듦
- 모델이 파라미터 초기화에 덜 민감하게 만들고 학습률 튜닝을 단순화 함

NIN 연결(1X1 합성곱)

- 채널이 많은 특성 맵을 얕은 특성 맵으로 매핑하는데 유용
- 소량의 파라미터로 비선형성을 추가로 주입할 수 있는 저렴한 방법



잔차 연결/ 잔차 블록

- 층이 100개 이상인 깊은 신경망을 가능하게 한 CNN의 중요한 트렌드
- skip 연결이라고도 부름
- 원본 행렬에 합성곱의 출력을 더하는 방법
- 출력 = conv(입력)+입력
- 합성곱 출력에 입력을 더하려면 같은 크기를 유지해야 함
    - 합성곱 수행 전에 패딩을 추가