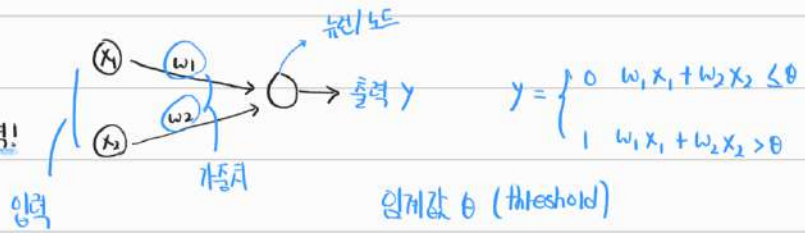
Chapter 1 퍼셉트론

퍼셉트론 (Perceptron) - 프랑크 로젠블라트 1957년에 고안 * 딥러닝의 기초

① 퍼셉트론이란?

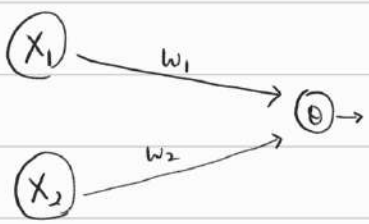
다수의 입력을 받아 하나의 출력

가중치가 존재 (w_1, w_2) → 가중치가 클수록 더 중요한 입력!



② AND 게이트와 퍼셉트론

x_1	x_2	AND
0	0	0
0	1	0
1	0	0
1	1	1



주어진 조건을 만족하는 w_1, w_2, θ 값이 존재?

$0 \cdot w_1 + 0 \cdot w_2 = 0 < \theta$

$1 \cdot w_1 + 0 \cdot w_2 = w_1 < \theta$

$0 \cdot w_1 + 1 \cdot w_2 = w_2 < \theta$

$1 \cdot w_1 + 1 \cdot w_2 = w_1 + w_2 > \theta$

⇒ (0.5, 0.5, 0.7) 등이 존재

NAND와 OR gate도 구현이 가능하다. → 지금은 사람이 입력 데이터를 보고 매개변수값 (가중치, 임계값) 설정

학습이란 이런 가중치의 값을 컴퓨터가 정하는 것!

③ 가중치와 편향 (bias)의 도입

$w_1x_1 + w_2x_2 \leq \theta \rightarrow w_1x_1 + w_2x_2 - \theta \leq 0 \Rightarrow \Sigma wx - \theta \leq 0$

$w_1x_1 + w_2x_2 > \theta$

편향 (bias)

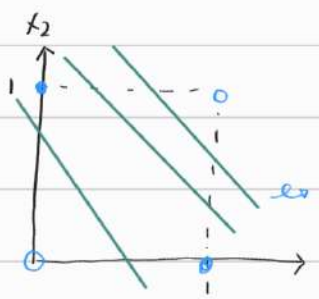
0을 넘으면 1 출력, 아니면 0 출력

편향 → 뉴런이 얼마나 쉽게 활성화 되는지를 결정 → 편향이 작을수록 작은 입력에도 활성화 (y=1)

④ 퍼셉트론의 한계

- XOR 게이트의 구현

x_1	x_2	XOR
0	0	0
1	0	1
0	1	1
1	1	0



$w_1x_1 + w_2x_2 + b$ 를 만족하는 w_1, w_2, b가 존재?

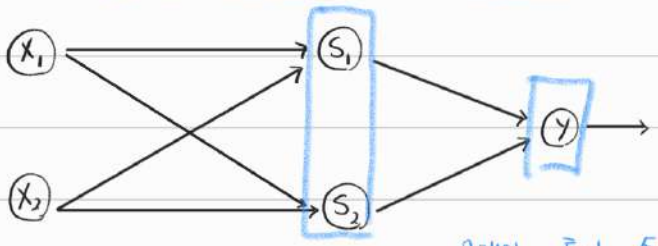
⇒ 한 개의 직선으로 분리 불가능

퍼셉트론은 단일 직선을 생성 → $w_1x_1 + w_2x_2 - b = 0$ ⇒ 이 직선 1개로 주어진 input을 분리 가능?

↳ 선형 / 비선형

ⓐ 다층 퍼셉트론

XOR 처럼 단일 퍼셉트론은 비선형문제 해결 불가능 → 퍼셉트론끼리 연결



2개의 층이 존재 ⇒ XOR (비선형) 해결가능

〈퍼셉트론 정리〉

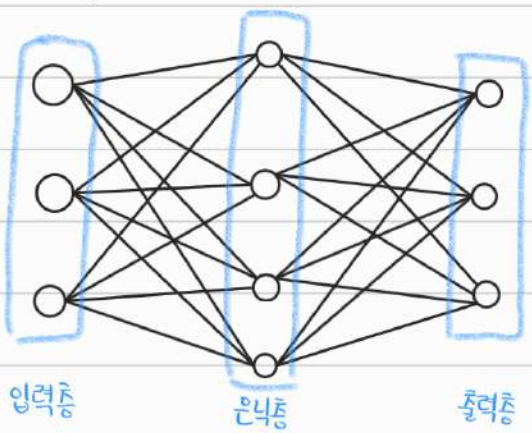
- 퍼셉트론은 입출력을 갖춘 알고리즘. 입력을 주면 주어진 결과에 따른 값 출력.
- 퍼셉트론은 가중치 와 편향 이라는 매개변수를 가짐
- 퍼셉트론으로 AND, OR와 같은 논리 회로 표현 가능
- XOR 게이트는 단층 퍼셉트론으로 표현불가 (단층 - 직선, 다층 - 비선형)
- 다층 퍼셉트론으로 이론 상 컴퓨터의 모든 연산 가능

chapter 2. 신경망

퍼셉트론에서의 매개변수 결정 → 사람이 직접

↳ 신경망을 이용하여 데이터로 부터 적절한 매개변수 값 학습

ⓐ 퍼셉트론에서 신경망으로



은닉층은 신경망 밖에서는 보이지 않음

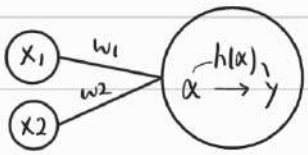
출력층 y의 동작 $y = \begin{cases} 0 & \Sigma wx + b \le 0 \\ 1 & \Sigma wx + b > 0 \end{cases}$ $\left(\begin{matrix} h(x) \le 0 \\ h(x) > 0 \end{matrix} \right)$

$h(\Sigma wx + b)$ $h(x) = \begin{cases} 0 \\ 1 \end{cases}$

· 활성화 함수의 등장

$h(x)$ → 활성화 함수

$\alpha = b + \Sigma w \cdot x \qquad h(\alpha) = \begin{cases} 0 \\ 1 \end{cases}$



⑥ 활성화 함수

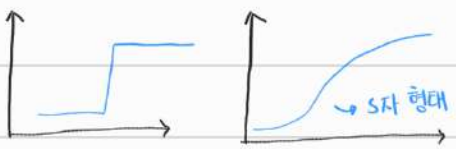
단일 퍼셉트론 → (0, 1) 만 가지는 계단함수를 활성화 함수로 사용

↳ 계단 함수에서 다른 함수로 변경하는 것이 신경망의 열쇠

시그모이드 (Sigmoid) 함수 - 신경망에서 자주 이용

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

계단함수와 시그모이드 함수



계단함수: 0을 기점으로 출력이 급격하게 변경

시그모이드 함수: 부드러운 곡선, 입력에 따라 출력 연속적 변화

↳ 둘 다 입력이 작을수록 출력도 작다 범위도 [0, 1] 만 가짐

- 비선형 함수

시그모이드, 계단함수 모두 비선형 함수 → 신경망에서는 비선형 함수만 활성화 함수로 사용

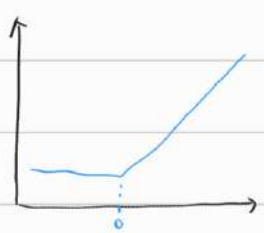
$h(x) = cx$ 의 경우 층이 3개이면 $h(h(h(x))) \rightarrow c(c(cx)) = c^3x$

$a = c^3$ 이면 결국 $h(x) = ax$ 의 형태 ⇒ 층을 깊이 두는 의미가 없다

다층 레이어를 구성한다면 반드시 비선형 함수 사용

- ReLu 함수

최근에 시그모이드보다 많이 사용



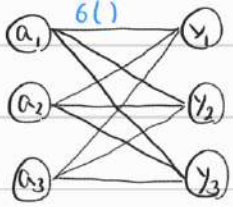
0이 넘으면 그대로 출력

0 이하면 0을 출력

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

- 출력층의 설계 - 회귀에는 항등함수, 분류에는 소프트맥스 함수 사용

- 소프트맥스 함수 : $y_k = \frac{\exp(a_k)}{\sum \exp(a_i)}$



- 소프트맥스 함수의 출력은 [0, 1] 사이의 실수
- 출력의 총합은 1 ↳ '확률'로 번역 가능

소프트맥스 함수 적용해도 원소의 대소관계는 변하지 않음

지수 연산에 드는 비용을 줄이고자 많이 생략함 → 단순 분류 문제에서는 불필요 (확률값이 필요없는 경우)

〈신경망 정리〉

- 신경망에서는 활성화 함수로 시그모이드나 ReLu 같은 매끄럽게 변화하는 함수 이용
- Numpy 다차원 배열 이용하여 신경망 효율적으로 구현 가능
- 출력층의 activation function으로 회귀(예측) 문제는 주로 항등 함수, 분류에서는 softmax 사용

Chapter 3. 신경망 학습

학습 - 훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득하는 것

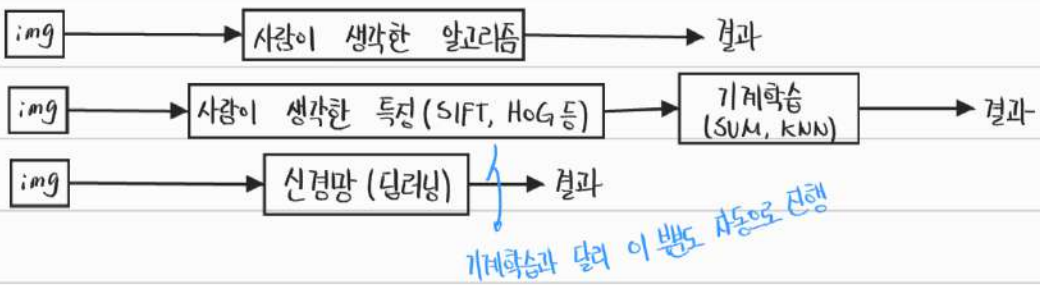
↳ 에서 필요한 것이 손실함수 (loss function)

↳ 손실함수의 값을 가급적 작게 만드는 것이 학습

학습의 핵심은 데이터로부터 가중치 자동 갱신!!

특정 데이터를 인식하는 알고리즘을 밑바닥부터 설계하는 것이 아니라 특징을 추출

↳ feature selection ↳ Input → feature 만들어낼 수 있는 변환기



데이터 — 훈련 데이터 - 매개변수 갱신
- 평가 데이터

- 손실함수

가중치 갱신을 위한 하나의 지표

오차제곱합 (SSE) $E = \frac{1}{2}\sum (y_i - \hat{y}_i)^2$

교차 엔트로피 오차 (CSE) 많이 사용 $E = -\sum y_i \log y_i$

- 손실함수를 정의하는 이유? ⇒ 미분을 하기 위해서

손실함수의 값을 가장 작게 하는 매개변수 값 탐색 → 매개변수의 미분 사용 (기울기)

가중치 매개변수의 손실 함수 미분 : 가중치 매개변수의 값을 변화시켰을 때, 손실함수가 어떻게 변하나

↳ 미분값 음수 ⇒ 가중치를 양으로 변화시켜 손실함수 값 감소

미분값 양수 ⇒ 가중치를 음으로 변화시켜 손실함수 값 감소

미분값 0 ⇒ 가중치 어느방향으로 움직여도 손실함수 변화 X

↳ 가중치 갱신 끝

'정확도' 대신 손실함수를 쓰는 이유 : 정확도의 경우 매개변수의 미분이 대부분 0이 되어버림

↳ 정확도의 경우 계단함수와 유사하게 동작 ⌐ → 세세한 매개변수 조정이 힘듦

손실함수의 미분을 이용해서 가중치 갱신 ⇒ 경사법 (gradient method)

$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

↳ eta 학습률 - 매개변수 값을 얼마나 갱신할 것인가?

신경망에서의 기울기

매개변수 $W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$ 미분 $\frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{23}} \end{pmatrix}$

신경망 학습 알고리즘

가중치, 편향을 학습 데이터를 이용하여 조정하는 과정이 학습

① 미니배치 - 학습 데이터 중 일부 무작위 추출 → 데이터를 미니배치로 무작위로 선정하기 때문에 확률적 경사 하강법 (SGD) 라고 부름

② 기울기 산출 - 손전파 or 손실함수의 기울기 계산

↳ 수치 미분 - 처음을 이용한 미분 구현 쉽고 시간 오래걸림

↳ 오차 역전파 - 구현 어렵지만 기울기를 빨리 구할 수 있음

③ 매개변수 갱신

④ 1 ~ 3단계 반복

Chapter 4. 오차역전파법

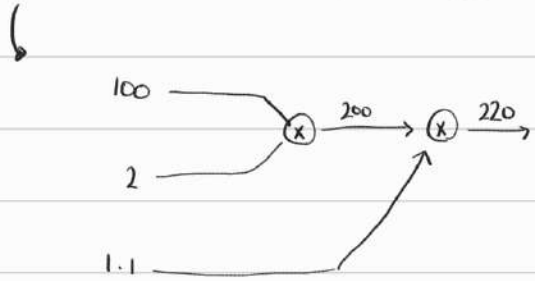
가중치 매개변수의 기울기 계산 ⇒ 수치 미분 → 단순하고 구현이 쉽지만 계산시간이 오래걸림

↳ 이걸 효율적으로 할 수 있는 방법 ⇒ 오차역전파법

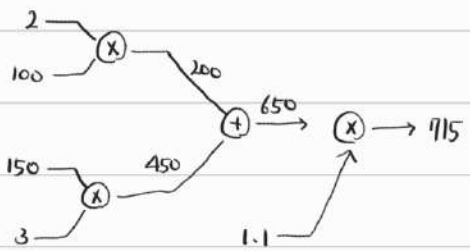
오차 역전파의 이해 — 수식 혹은 계산 그래프 → 계산 과정을 시각적으로 표현

ex) 100원 짜리 사과 2개, 소비세 10%, 총 지불 금액

100 →(×2) 200 →(1.1) 220 → 220



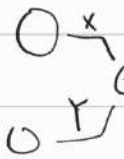
<조금 더 복잡한 계산 그래프>



→ 순전파

← 역전파

'국소적 계산'을 이용해 최종 결과 도출



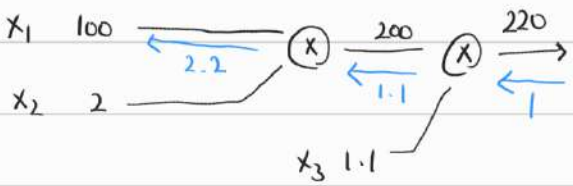
→ x+y

x, y 이전에 어떤 계산이 있던 상관없음

현 단계에서는 x, y의 덧셈만 계산

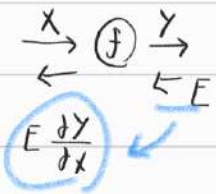
→ 아무리 복잡해도 각 노드에서는 단순한 문제를 계산

역전파



→ x_1이 1오르면 전체 금액이 2.2만큼 증가

연쇄법칙 (Chain rule)



$y = f(x)$

역전파의 계산 순서 : 앞 노드의 값 E를 전달받아서 국소 미분값 곱하고 뒤로 전달

↳ $z = (x+y)^2$ 과 같은 합성함수의 미분 ⇒ 각 함수 미분의 곱으로 나타낼 수 있다 (연쇄법칙)

$z = t^2$, $t = x+y$ → $\frac{dz}{dx} = \frac{dz}{dt} \times \frac{dt}{dx}$ → $2t \times 1 = 2(x+y)$

미분값이 chain-rule에 의해 전파되면서 기울기 빠르게 계산

역전파의 과정 → 계산 그래프로 이해!

Chapter 5. 학습 관련 기술들

매개변수 갱신

손실함수 값을 최소화 하는 매개변수 값 찾기 ⇒ 최적화 → 매우 어려움

확률적 경사 하강법 (SGD) 주로 이용

↳ 기울기가 가장 큰 방향으로 이동

비등방성 함수 에서는 탐색 경로가 비효율적 → 지그재그로 탐색

↳ 방향에 따라 성질이 달라지는 함수

- 모멘텀 (Momentum) - 운동량

모멘텀 $v = \alpha v \boxed{-\eta \frac{\partial L}{\partial W}}$ $W \leftarrow W + v$ → '속도'에 해당

αv: momentum 설정 시 값 (0.9 등)

$-\eta \frac{\partial L}{\partial W}$: 현재 가중치 갱신량

기울기만큼 가속도를 받는다

- Adagrad

학습률이 매우 중요 ┌ η가 너무 작으면 학습 시간이 길어짐
└ η가 너무 크면 발산하여 학습이 이루어지지 않음

학습률을 효과적으로 정하기 ⇒ 학습률 감소 → 처음에는 학습률을 크게 하다가 점차 감소

Adagrad → 각 매개변수에 맞게 학습률 감소 가장 쉬운 방법은 전체 학습률 다 내리기

$$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

↓ 기울기가 큰 매개변수는 학습률이 작아진다

· Adam

momentum + Adagrad → 정확한 내용은 아니지만 이 정도로 이해

완벽한 정답의 갱신기법은 없음 ⇒ 상황과 문제에 따라 다른 값을 보임

↳ SGD가 많이 쓰였지만 Adam도 많이 쓰는 추세

가중치의 초깃값

가중치 초깃값에 따라 신경망 학습이 좌우될 정도로 중요!

가중치가 0으로 모두 같다면 ⇒ 모든 가중치 갱신 값이 같아짐

↳ 가중치와 은닉층을 여러 개 가지는 의미가 없어짐

층이 여러 개인 이점을 활용하려면 은닉층의 활성화값(출력값)이 고루 분포되어야 한다

↳ 이걸 이루기 위한 가중치 초깃값이 Xavier 초깃값

Xavier를 이용하여 각 층의 출력분포 개선
↳ 선형에 특화

- Sigmoid는 층이 깊어질수록 쏘그라짐 (출력 0,1에서 대칭일수록 좋은 활성화함수)
- tanh를 사용하면 이 문제 해결

ReLu에 특화 ⇒ He 초깃값

〈현재 모범 사례〉 Sigmoid / tanh : Xavier 초깃값 ReLu : He 초깃값

배치 정규화

가중치의 초깃값이 적절하면 활성화 분포가 퍼지면서 학습이 원활히 수행
↳ 각 층이 활성화를 적당히 퍼뜨리도록 "강제" ⇒ 배치정규화

배치 정규화 - 2015년에 고안

- 학습 진행을 빨리 할 수 있음
- 초깃값에 큰 영향을 받지 않는다
- 오버피팅을 억제한다 (드롭아웃 등의 필요성 감소)

Affine → Batch norm → ReLu

↳ 미니배치를 단위로 정규화 (평균 0, 분산 1)

$\mu_B = \frac{1}{m}\sum x_i$

$\sigma_B^2 = \frac{1}{m}\sum (x-\mu_B)^2$

$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma_B^2 + \varepsilon}}$

Batch norm → 미니 배치 입력 데이터를 정규화하는 계층

↳ 이걸 활성화 함수 앞/뒤에 배치

배치 정규화 계층마다 정규화된 데이터에 확대(scale) / 이동(shift) 수행 $(y_i \leftarrow \gamma \hat{x}_i + \beta)$ (γ: 확대, β: 이동)

오버피팅 제어

(매개변수가 많고 표현력이 높은 모델, 학습 데이터가 적은 경우) → 오버피팅이 잘 일어남

- 가중치 감소 (weight decay)

  학습 과정에서 큰 가중치에 대해 페널티 부과

  ex) 가중치의 제곱 norm (L2 norm)을 손실함수에 더하기

  ↳ 가중치가 커질수록 손실함수도 커진다 $\frac{1}{2}\lambda W^2$ (λ: 하이퍼 파라미터)

- 드롭아웃 (dropout)

  뉴런을 임의로 삭제하는 학습 기법 (학습 시에 임의로 삭제, 평가 시에 모든 뉴런 이용 → 학습 시에 삭제한 비율 곱하여 출력)

적절한 하이퍼 파라미터 값 찾기

각 층의 뉴런 수, 배치 크기, 학습률 등의 다양한 하이퍼 파라미터 존재

- 하이퍼 파라미터 값 설정 → 검증 데이터 사용!

  ↳ 평가 데이터를 사용하면 하이퍼 파라미터가 평가 데이터에 적응

- 하이퍼 파라미터 최적화

  ↳ '최적 값'이 존재하는 범위를 조금씩 줄여 나가기 → 대략적인 범위 설정 후 무작위로 하이퍼 파라미터 설정, 그 값으로 정확도 평가 (규칙적인 탐색보다 무작위 탐색이 효과적이라고 알려져 있음)

  ↳ 0.001 에서 1000 사이 ($10^{-3} \sim 10^{3}$) 까지 10의 거듭 제곱 단위 (로그스케일)

① 하이퍼 파라미터 값의 범위 지정

② 설정된 범위에서 무작위 값 추출

③ 추출된 값으로 학습 검증 데이터로 정확도 평가 (epoch은 작게)

④ 위 단계 반복

정리

1. 매개변수 갱신법: SGD, Adagrad, Adam 등
2. 가중치의 초깃값 설정이 매우 중요 → Xavier, He 등
3. 배치 정규화 이용하면 학습 속도 빨라지고 초깃값 영향 작다
4. 오버피팅 억제 → 가중치 감소, 드롭 아웃
5. 하이퍼 파라미터 탐색 → 범위 지정 후 감소

Chapter 6. 합성곱 신경망 (CNN)

합성곱 신경망 (Convolutional neural network; CNN)

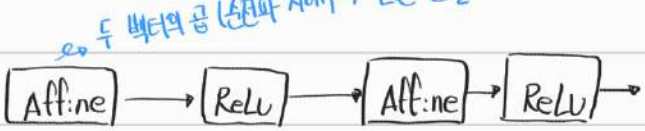
이미지 인식, 음성 인식 등에 사용

전체 구조

기본적인 구조는 일반적인 신경망 (MLP)와 유사

↳ 여기에 합성곱 계층 (Convolutional layer)와 풀링 계층 (Pooling layer) 추가

기존 신경망)



합성곱 신경망)

합성곱 계층

- 완전연결 계층 (Affine 계층)의 문제 ⇒ 데이터의 형상이 무시됨
  - ↳ 이미지같은 3차원 (R, G, B 계층) 정보가 중요한 경우 큰 단점

CNN은 데이터의 형상 유지 CNN에서의 입출력 데이터 : feature map

- 합성곱 연산

$1\times2 + 2\times0 + 3\times1 + \cdots + 3\times1 + 0\times0 + 1\times2 = 15$ (단일 곱셈 누산)

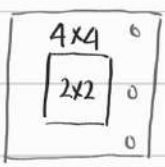
필터의 윈도우를 일정 간격 이동해가며 전체 데이터에 적용 (fused-multiply-add FMA)

필터의 값들이 신경망에서의 매개변수

편향 (bias) 적용 시 전체 계산결과 $\begin{pmatrix}15 & 16\\ 6 & 15\end{pmatrix}$ 에 한번에 적용 → bias는 항상 (1,1)의 크기

- 패딩 (padding)

입력 데이터 주변을 0으로 채우는 것



→ 입력은 2x2

padding=1을 통해 4x4으로

padding을 하는 이유 : 출력 크기 조정

padding 없이 계속 합성곱 적용할 경우 데이터 크기가 계속 줄어들어 결국 1이 됨

⇒ 입력 데이터의 공간적 크기 고정한 채로 다음 계층에 전달가능

- 스트라이드 (stride)

필터를 적용하는 위치의 간격 (윈도우가 얼마만큼 이동하는가)

- padding, stride, 출력 크기의 관계

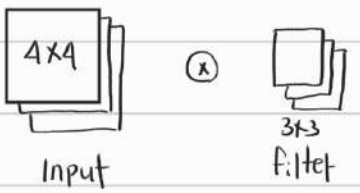
입력 크기 : (H, W) 필터 크기 : (FH, FW) 출력 크기 : (OH, OW) 패딩 : P 스트라이드 : S

$$OH = \frac{H + 2P - FH}{S} + 1 \qquad OW = \frac{W + 2P - FW}{S} + 1$$

단, 둘 다 모두 정수

- 다차원에 적용

각 채널에 필터 적용 후 모두 더하기



데이터의 채널 수 = 필터의 채널 수

→ 모두 더해서 채널을 없앨수도 있고 유지할 수도 있음

풀링 계층

각 차원의 크기를 줄여주는 계층 (가로·세로 방향의 공간 크기 감소)

Max, mean, pow 등의 방법이 존재

↳ 윈도우 내의 가장 큰 값을 선택

pooling의 특징

① 학습해야할 매개변수가 없다 ② 채널 수가 변하지 않는다 ③ 입력의 변화에 강건하다

정리

① CNN은 Affine 계층 대신에 Convolutional layer와 pooling layer 사용

② CNN을 시각화하면 층이 깊어질수록 고급 정보 추출 → 추상화된 정보

Chapter 7 딥러닝

층을 깊게 하는 이유 → 이론적인 (정확한 수학적 근거) 근거는 부족하지만 성능이 좋음

↳ 층의 깊이에 비례해 정확도 향상

층이 깊어질수록 매개변수 수가 줄어듦 ⇒ 적은 수의 변수로 같은 (혹은 그 이상의) 표현력을 낼 수 있음

ex) 5x5 합성곱 연산 : 매개변수 25 (5x5)
3x3 합성곱 2번 : 매개변수 18 (3x3x2)
→ 층이 깊어질수록 차이 심해짐

학습의 효율성도 더 좋아짐 → 층을 깊게 하여 학습 데이터 양을 줄여 학습을 고속으로 수행 가능

↳ '얕은' 층에서는 특징을 파악하기 위해 많은 수의 데이터가 필요한 것을 해결