

# Deep Learning

from Scratch

---

밀바닥부터 시작하는 딥러닝

### | 표지 설명 |



표지 물고기는 쇼뱅이다(학명: *Sebastiscus mammoratus*). 쇼뱅이는 페르카목 쇼뱅이과에 속한다. 몸길이는 20cm 정도이며 몸은 쏘기과와 비슷하나 등지느러미 가시가 열두 개인 점으로 구별한다. 몸 뒷쪽은 서식하는 장소에 따라 다양하여 일반적으로 연안의 것은 흑갈색이고, 깊은 곳의 것은 흰빛을 띤다. 몸 옆구리에는 다섯 줄의 불규칙한 흑갈색 가로띠가 있다. 양턱은 거의 같은 길이나 아레타이 약간 짧다. 머리의 가시는 길고 날카로우며 두 눈 사이는 깊이 평어져 있고 용기 연(縫)의 무풀은 머리가사에서 끌난다. 비늘은 작은 빛바늘이며 안성어로 연안의 암초부에 서식한다. 태생이며 12~4월에 새끼를 낳는다.

표지 그림은 라이데커의 'Royal Natural History'에서 가져왔고, 설명은 위키백과에서 발췌했다.

## 밀비닥부터 시작하는 딥러닝

프리씨으로 익히는 딥러닝 이론과 구현

초판 1쇄 발행 2017년 1월 3월

초판 12쇄 발행 2020년 10월 21일

저운이 사이토 고기 / 옮긴이 개요만사 / 편번이 김태현

펴낸곳 한빛미디어주 // 주소 서울시 서대문구 연희로2길 62 흰빛미디어(주) [T출판부]

전화 02-325-5544 / 팩스 02-336-7124

등록 1999년 6월 24일 제25100-2017-000058호 / ISBN 978-89-6848-463-6 93000

총괄 전성우 / 책임편집 홍성신 / 기획 · 편집 이복연 / 진행 이윤지

디자인 표시 · 내지 역동 일 조판 이경숙

영업 김형진, 김진불, 조유미 / 마케팅 박상웅, 송경석, 조수현, 이형은, 고광일 / 제작 박성우, 김정우

0 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 흰빛미디어(주)의 흰빛미디어(주)의 흰빛미디어(주)에 표시되어 있습니다.

이 책의 저작권은 오라일리재팬과 흰빛미디어(주)에 있습니다.

저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

Copyright © Hanbit Media, Inc. 2017

Authorized translation of the Japanese edition of 'Deep Learning from Scratch' © 2016 O'Reilly Japan, Inc. This translation is published and sold by permission of O'Reilly Japan, Inc., the owner of all rights to publish and sell the same.

한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다!

지금 하지 않으면 할 수 없는 일이 있습니다.

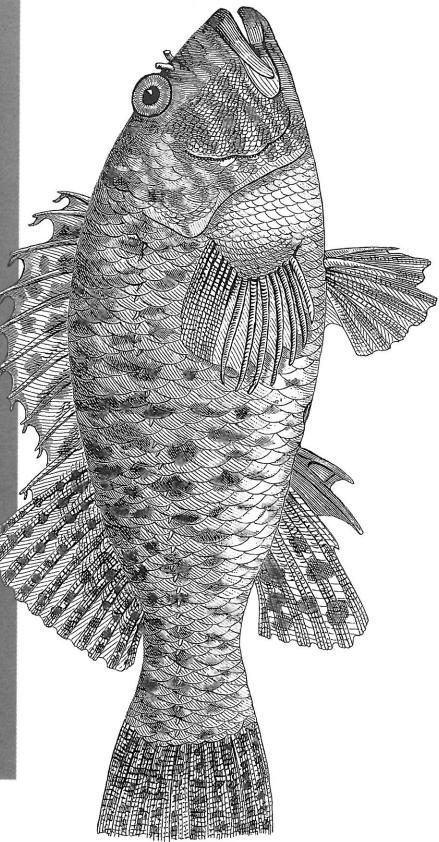
차으로 펴내고 싶은 이디어나 원고를 메일 ([writer@hanbit.co.kr](mailto:writer@hanbit.co.kr))로 보내주세요.

한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다!

# Deep Learning

from Scratch

일바닥부터 시작하는 딥러닝



O'REILLY®  한빛미디어

Hanbit Media, Inc.

## 지은이 · 옮긴이 소개

지은이 **시이|토 고카**(斎藤 康毅)

1984년 나가사키 현 쓰시마 태생. 도쿄공업대학교 공학부를 졸업하고 도쿄대학원 학제정보학부 석사 과정을 수료했다. 현재는 기업에서 컴퓨터 비전과 기계학습 관련 연구·개발에 매진하고 있다. 오리얼리재팬에서『실천 파이썬 3』,『컴퓨터 시스템의 이론과 구현』,『실천 기계학습 스텝』 등을 번역했다.

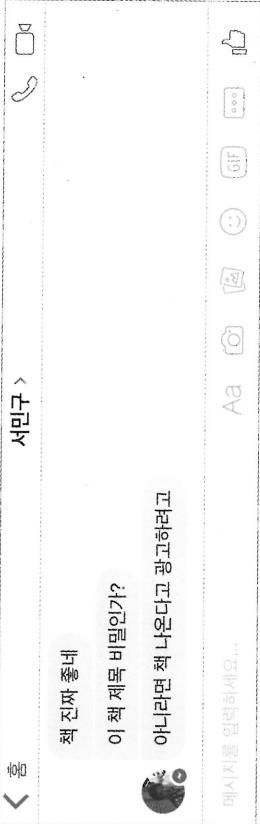
옮긴이 **개앞풀시(이복연)** wegra.lee@gmail.com

고려대학교 컴퓨터학과를 졸업하고 삼성소프트웨어멤버십을 거쳐, 삼성전자 소프트웨어센터와 미디어솔루션센터에서 자바 개발자, 멤버인, 바다 플랫폼, 첫온 메신저 서비스 등을 개발했다. 주 업무 외에 분산 빌드, 지속적 통합, 앱 수명주기 관리 도구, 애자일 도입 등 동료 개발자들에게 실질적인 도움을 주는 일에 적극적이었다. 그 후 창업전선에 뛰어들어 소셜 서비스, 금융 가래 프레임워크 등을 개발하다가, 무슨 바람이 불어 차인자 책을 만들겠다며 기획·편집자(차장 Wisdom Compiler)로 변신했다.

『O|피티브 자바』, 3판(인사이트, 2018),『Effective Unit Testing(한빛미디어, 2013), JUnit 인 액션』(인사이트, 2011)을 번역했다.

〈개발자의 앞길에 맨해 시전〉, 줄어서 ‘개앞풀시’는 역지가 어려서부터 생각한 후회 양성의 품을 조금 독특한 방식으로 일찍 실행에 옮긴 것이다. 현재 모습은 게임, 서버, 웹 등 주요 직군별 개발자에게 꼭 필요한 기술과 역량을 안내하는 책들을 로드맵 형태로 정리한지도다. 필요할 때 바로 구해볼 수 있도록 판매 중인 도서만을 다룬다.

- 페이스북 : <https://facebook.com/dev.loadmap>
- 로드맵 모음 : <https://www.mindmeister.com/ko/users/channel/865528>
- 스키아이넷도 텀블링부터 : [https://mindmeister.com/812276967/\\_](https://mindmeister.com/812276967/_)



신경망과 딥러닝의 기본을 밀바닥부터 만들어보면서 그 개념을 어렵지 않게 이해할 수 있었습니다. 이는 텐서플로 등의 딥러닝 프레임워크를 사용할 때도 진가를 발휘하리라 생각합니다. 딥러닝 공부를 시작하시는 모든 분께 추천합니다.

서민구, 「R을 이용한 데이터 처리&분석 실무」 저자 |  
구글코리아 소프트웨어 엔지니어

탄탄한 구성으로 신경망과 딥러닝 전반을 쉽게 설명해주고, 특히 역전파는 제가 본 교재 중 가장 쉽게 서술했습니다. 복잡한 수식이나 난해한 설명이 아닌, 이야기를 하듯 독자를 이해시키려 한 부분이 가장 감명 깊었습니다. 신경망(딥러닝)을 공부하며 역전파에서 어려움을 겪은 분들께 강력하게 권해드립니다.

안홍철, 에셋플러스자산운용 자산운용사

비터리딩을 위해 처음 받은 원고부터 거의 완성된 책이나 다름없어서 읽기 편했습니다. 독자들이 쉽고 빠르게 맛보고 입문할 수 있을 것입니다.

박상은, 「9가지 사례로 익히는 고급 스파크 분석」 저자 |  
에스코어

지금 이 책의 원서는 일본 대형 서점의 매대 중앙을 차지하고 TV에도 소개될 정도로 인기입니다. 공학서적이 이렇게 주목받는 것은 인공지능에 관한 일반인의 관심이 그만큼 높아졌다는 방증이겠지요. 한국에서도 이 책이 인공지능 보급에 중요한 기폭제가 되길 기대합니다.

최성훈, Tomomi Research Inc.

2016년 11월 29일 미국의사협회지 JAMA에 실린 <딥러닝을 이용한 당뇨성 망막병증 진단> 논문은 인공지능이 바꾸는 안과 미래의 시작으로 평가받습니다. 수학과 프로그래밍 언어에 익숙하지 못한 의료 관련 분들에게도 이 책은 제목처럼 ‘필비닥부터’ 딥러닝을 이해시켜줄 것입니다.

임형태, 안과전문의

일본인 특유의 꿈꿉함이 느껴지는 너무 좋은 내용이었습니다. 개인적으로 이 책처럼 처음부터 하나 하나 개발하는 것을 좋아합니다. 빨리 정식 출간되어 내년 사내교육 때 이 내용을 활용하고 싶을 정도로 맘에 듭니다.

이성훈, ‘캐리스 코리아’와 한국 스파크 사용자 모임 운영자  
삼성생명 DA Lab

저와 같은 어린 학생들이 직업을 찾을 즈음에 인공지능이 사회의 모든 부분을 변화시키는 혁명이 되어 있을 거라고 합니다. 그래서 이제 겨우 중 2인 저도 인공지능에 관심이 많았는데, 마침 리뷰어로 접한 이 책은 아직 파이썬에 낯선 저도 쉽게 따라 배울 수 있었습니다.

김경수, 파주한빛중학교 2학년 학생

멋진 책입니다! 딥러닝이 궁금하여 이 책을 접어 드셨다면 시답잖은 옮긴이의 말에 시간 낭비 마시고 곧장 본문으로 뛰어드세요. 쉽고 재밌고 유익한 여행이 될 겁니다.

제 조언을 무시하고 시간을 낭비 중이신 독자께...

번역을 마무리하고 있는 지금, 일본에서 베타리딩을 해주신 최성훈 님께서 현지 소식을 전해주셨습니다. 이 책의 원서가 일본 대형 서점의 매대 중앙을 당당히 점하는 모습이 낯설지 않고 최근엔 TV에서도 책 내용을 소개했다고 합니다. 그동안 아마존재팬 사이트에서 무려 '종합' 베스트 50위 근방을 오르내리는 모습에 대단하다 싶었는데, 오프라인에서도 그 인기가 정말 뜨겁다고 하는군요. 그래서 이 좋은 책을 여러분께 하루라도 빨리 선보이고자 오늘도 달립니다.

딥러닝을 필두로 한 인공지능의 열기는 우리나라도 다르지 않습니다. 여러분이 더 잘 느끼시겠지요. 저는 불과 작년까지만 해도 인공지능이 이렇게 급작스럽게 일반 대중의 관심까지 사로잡을 줄은 상상도 못 했고, 2000년대 초 인공지능 수업을 들을 때는 아득히 먼 미래 이야기란 느낌이었습니다. 우리 인간의 뇌를 구성하는 1천억 뉴런의 동시다발적 상호작용을 단순한 인공 뉴런과 체현된 컴퓨터 성능으로 과연 어디까지 흥내 낼 수 있을까 하는 의구심이었죠.

하지만 제가 늙어가는 사이에도 수많은 훌륭한 선구자께서 새로운 이론을 발전시키고 더 좋은 하드웨어와 소프트웨어 기반을 꾸준히 다져주셔서 '드디어' 꽃봉오리를 맺었습니다. 이 봉오리를 기꺼이 꽂으로 피워낼자는 이제 독자 여러분의 몫입니다.

인공지능은 터 이상 개발자의 전유물이 아닙니다. 이 책을 읽어보면 아시겠지만 디단한 프로그래밍 능력이나 특별한 전공 지식은 필요 없습니다. 어쩌면 고등학교를갓 졸업해 수학 감이 살아 있는 분들이 더 유리할 수도 있습니다. 마치 '생활 데이터 과학'처럼 우리 일상에서 누구나 손쉽게 활용할

수 있는 '생활 인공지능'도 곧 퍼지지 않을까 기대해봅니다.

제법 많은 평일 저녁과 주말을 책 번역에 바치면서 내가 이리라고 번역을 시작했나 자괴감 들고 괴로울 때는 다행히 아직 없었습니다. 그리고 이 책이 여러분을 딥러닝의 세계로 친절히 모신 첫 책이 있다는 이야기가 종종 들려온다면 앞으로도 영원히 없을 것입니다. 제가 쓰진 않았지만 제 책이라 생각하고 성심껏 A/S할 테니, 책 내용에 대해 궁금한 것은 언제든 문의해주세요.

마지막으로, 지금도 열심히 딥러닝을 서비스에 적용하고 계신 각계각층의 현업 개발자부터 꿈많은 중학생 친구까지, 부족한 시간 조기어 제 원고를 검토하고 다음과주신 많은 리뷰에게 진심으로 감사드립니다.

## 스카이넷도 딥러닝부터

번외로, 이 책 다음으로 어떤 책을 읽어야 할지 모르겠거나, 지금 나와 있는 다른 책들은 뭐가 있는지 궁금한 분들을 위해 개요맵시 인공지능/딥러닝 편을 준비했습니다.

- [https://mindmeister.com/812276967/\\_](https://mindmeister.com/812276967/_)

대학교내 대학원 때 배우는 교재 형태는 가능한 한 빼고, 현업 개발자에 유용한 책들을 담았습니다. 그래서 초기엔 많이 부실할 것입니다만, 곧 수많은 출판사에서 더 많은 인공지능 관련 책을 내어주리라 믿습니다. 좋은 책이 나오면 지속해서 추가할 테니 종종 들려주세요.

## 번역 용어표

외국 문학과 논문 등으로 공부하신 분은 번역어가 낯설 수 있어서 시중의 다른 책들도 참고해 표로 정리해봤습니다. 쭉~ 보시면 우리말 책들이 사용하는 용어에 어느 정도는 익숙해지실 겁니다.

- <https://goo.gl/47Djv1>

SF영화 같은 세계가 현실로 바뀌었습니다. 인공지능이 장기와 체스 챔피언에 승리했고, 최근에는 바둑에서도 이세돌 9단에게 승리해 전 세계적으로 큰 화제가 되었습니다. 스마트폰은 사람의 말을 이해하고 통화 중 실시간 ‘기계 통역’도 가능합니다. 카메라와 센서를 탑재한 ‘무인 자동차’에서 보듯 사람의 생명을 지키는 차율 주행 자동차도 곧 실용화될 조짐을 보입니다. 이렇게 주위를 둘러보면 사람만 못하다고 생각한 분야의 작업을 인공지능은 실수 없이 처리하고, 어떤 경우엔 사람보다 끝나고 합니다. 우리의 세계는 인공지능의 발전으로 새로운 세계로 거듭나고 있습니다.

이런 발전과 눈부신 세계의 이면에서는 ‘딥러닝(심층학습)’이라는 기술이 중요한 역할을 담당하고 있습니다. 세계의 연구자들은 딥러닝을 혁신적 기술로 평고, 어떤 이는 수십 년 만에 한번 나오는 혁신이라며 칭찬을 아끼지 않습니다. 사실 이 딥러닝이라는 말은 연구자나 엔지니어 사이에서뿐만 아니라 일반에도 알려진 단어로, 뉴스나 잡지에 소개될 정도로 시선을 끌고 있습니다.

이 책은 이처럼 핫한 학제거리인 ‘딥러닝’을 다룹니다. 딥러닝 기술을 최대한 깊이(deep) 이해하는 것을 목적으로 ‘밑바닥부터 만든다는 콘셉트를 내걸고 있습니다.

이 책의 특징은 ‘만드는 과정’을 통해서 딥러닝의 본질에 접근한다는 데 있습니다. 딥러닝 프로그램을 구현하는 과정에서 필요한 기술을 (가능한 한) 생략하지 않고 설명합니다. 또, 실제로 작동하는 프로그램도 제공하여 독자가 직접 다양한 실험을 해볼 수 있도록 배려했습니다.

딥러닝을 만들기까지는 많은 시련이 있어 깊지 않은 시간이 필요하겠지만, 그만큼 배우는 것도 많을 것입니다. 그리고 내 손으로 직접 만들어본다는 건 즐겁고 가슴 설리는 일이죠. 이 책과 함께 딥러닝에서 사용하는 기술과 친해지고, 그 즐거움을 느낄 수 있기를 바랍니다.

자, 딥러닝은 이미 전 세계의 모든 장소에서 살아 움직이고 있습니다. 누구 손에나 들려 있는 스마트폰에서도 딥러닝이 작동하고 있습니다. 차율 주행 자동차에서도, 웹 서비스를 떠받드는 서버에서도 딥러닝이 활발히 돌아가고 있습니다. 많은 사람이 알지 못하는 곳에서 오늘도 딥러닝은 조용히 춤을 춥니다. 그리고 앞으로도 딥러닝의 춤은 열기를 더할 것입니다. 이 책으로 딥러닝에 얹힌 기술을 이해하고, 딥러닝의 춤에 매료되길 바랍니다.

## 0 | 책의 특징

이 책은 ‘딥러닝’에 대한 책입니다. 딥러닝을 이해하는 데 필요한 지식을 기초부터 하나하나 차례로 설명합니다. 딥러닝이 무엇인지, 어떤 특징이 있는지, 어떤 원리로 동작하는지 등을 최대한 쉬운 말로 설명합니다. 간단한 기술 개요에서 한 걸음 더 내디뎌 깊이 이해할 수 있도록 했습니다. 이것이 이 책의 특징이자 장점입니다.

그럼, 딥러닝을 ‘더 깊이’ 이해하려면 어떻게 하면 좋을까요? 가장 좋은 방법은 당연히 실제로 만들 어보는 것입니다. 실제로 움직이는 프로그램을 밀비막부터 만들어 그 소스 코드를 읽고 고민해보는 과정은 어떠한 기술을 제대로 이해하는 데 아주 중요합니다. 여기서 ‘밀비막부터’란 함은 되도록이 면 이미 만들어진 외부 라이브러리나 도구 등에 기대지 않는다는 뜻입니다. 즉, 이 책의 목적은 속 모를 블랙박스는 되도록 사용하지 않고 우리가 이해할 수 있는 최소한의 지식에서 출발하여 최첨단의 딥러닝을 이끄는 것입니다. 그리고 그 만드는 과정을 통해서 딥러닝을 더 깊이 이해하는 것이죠.

자동차 관련 책에 비유한다면, 이 책은 운전교습 책이 아닙니다. 운전하는 방법이 아니라, 자동차의 원리를 이해하는 것을 목표로 합니다. 자동차의 구조를 이해하기 위해 보닛을 열고 조목조목 부품을 만져보고 확인하고 움직여보겠습니다. 그리고 핵심을 최대한 간결한 형태로 추출하여 그 차의 프라모델을 조립해볼 겁니다. 이 책은 자동차를 만드는 과정을 통해서, 그리고 직접 자동차를 만들어본다는 생각으로 자동차 관련 기술과 친해지게 해드립니다.

또한, 이 책에서는 딥러닝을 만드는 데 파이썬이라는 프로그래밍 언어를 사용합니다. 파이썬은 인기 도 많고 처음 접하는 사람도 쉽게 이해할 수 있답니다. 특히 프로토타입을 만드는 데 적합하여, 머릿속에 떠오른 아이디어를 즉시 시험하고 결과를 보면서 여러 실험을 해볼 수 있습니다. 이 책에서는 딥러닝에 대한 이론적인 설명과 함께 파이썬으로 프로그램을 구현하고 다양한 실험을 해볼 것입니다.

**NOTE** 수식과 이론 설명만으로 어려울 때는 소스 코드를 알고 움직여보면 명확하게 이해되며 경우가 많습니다. 어려운 수식이라도 이를 구현한 소스 코드를 읽어보면 그 동작 흐름이 고려하는 것을 많은 사람이 경험했을 것입니다. 이 책은 실제 동작하는 코드 수준에서 딥러닝을 이해하려는 엔지니어링에 중점을 두었습니다. 수식이 많이 나오지만, 그만큼 프로그래머를 위한 소스 코드도 많이 등장합니다.

## 누구를 위한 책인가?

이 책은 동작하는 딥러닝을 직접 구현하며 깊이 이해할 수 있도록 구성했습니다. 다음을 보면 누구를 위한 책인지 더 명확해질 것입니다.

- 외부 리소스라는 최소한만 이용하고 파이썬을 사용해 딥러닝 프로그램을 처음부터 구현합니다.
- 파이썬이 처음인 사람도 이해할 수 있도록 파이썬 사용법도 간략히 설명합니다.
- 실제 동작하는 파이썬 코드와 독자가 직접 실험할 수 있는 학습 환경을 제공합니다.
- 간단한 기계학습 문제부터 시작하여 궁극에는 이미지를 정확하게 인식하는 시스템을 구현합니다.
- 딥러닝과 신경망 이론을 알기 쉽게 설명합니다.
- 오차역전파법과 합성곱 연산 등 복잡해 보이는 기술을 구현 수준에서 이해할 수 있도록 설명합니다.
- 하이퍼파라미터 결정 방식, 기종치 조건값 등 딥러닝을 활용하는 데 도움이 되는 실천적인 기술을 소개합니다.
- 배치 정규화, 드롭아웃, Adam 같은 최근 트렌드를 설명하고 구현해봅니다.
- 딥러닝이 왜 뛰어난지, 총이 깊어지면 왜 정확도가 높아지는지, 은닉층이 왜 중요한지와 같은 '왜'에 관한 문제도 다룹니다.
- 지을 주제, 이미지 생성, 강화학습 등, 딥러닝을 응용한 예를 소개합니다.

## 누구를 위한 책이 아닌가?

“누구를 위한 책이 아닌가”도 중요합니다. 다음은 이 책에서 주목하지 않는 주제입니다.

- 딥러닝 분야의 최신 연구에 대해서는 자세히 다루지 않습니다.
- 카페Caffe, 텐서플로TensorFlow, 체이너Chainer 등의 딥러닝 프레임워크 사용법은 설명하지 않습니다.

- 딥러닝, 특히 신경망에 관한 아주 상세한 이론까지는 담지 않았습니다.
- 딥러닝의 정확도를 높이기 위한 투닝은 자세히 설명하지 않습니다.
- 딥러닝 성능을 높여주는 GPU 기술은 구체적으로 다루지 않습니다.
- 주로 이미지 인식을 다룹니다. 자연어 처리, 음성 인식 등의 시례는 다루지 않습니다.

이처럼 이 책은 최근 연구나 이론적인 세부 설명은 다루지 않습니다. 그러나 이 책을 읽고 나면, 그 다음 단계로 최신 논문과 신경망 관련 기술서를 읽기가 한결 편해질 것입니다.

**WARNING** 이 책은 이미지 인식을 주제로 합니다. 주로 딥러닝으로 이미지를 인식하는 데 필요한 기술을 배웁니다. 자연어 처리, 음성 인식 등은 이 책의 대상이 아닙니다.

## 이렇게 읽으세요

새로운 지식을 배울 때 설명만 들어서는 석연치 않거나 금방 잊어버리게 됩니다. 배운이 불여일전 이듯, 새로운 것을 배울 때는 무엇보다 '실습'이 중요합니다. 이 책의 각 장은 주제 하나를 설명한 후 그것을 실습할 수 있도록 꾸몄습니다. 즉, 실행되는 소스 코드를 준비했습니다.

이 책은 파이썬 소스 코드를 제공합니다. 여러분이 실제로 움직여볼 수 있는 코드입니다. 소스 코드를 읽으면서 스스로 생각하고 자신이 생각한 것을 반영하고 실험하다 보면 확실히 자기 것으로 만들 수 있습니다. 여러 실험을 해보면서 겪는 시행착오 역시 큰 도움이 될 것입니다.

이 책은 '이론 설명'과 '파이썬 구현 코드'라는 두 트랙으로 진행합니다. 그래서 프로그래밍할 수 있는 환경을 준비할 것을 권장합니다. 윈도우, 맥, 리눅스 등 어떤 환경도 문제없습니다. 파이썬 설치와 사용법은 "1장 헬로 파이썬"에서 설명하겠습니다. 이 책에서 사용하는 프로그램은 다음의 것허브 저작소에서 확인하실 수 있습니다.

- <https://github.com/WegraLee/deep-learning-from-scratch>

## 그럼, 시작해보죠!

서론은 이것으로 끝입니다. 여기까지의 설명을 읽고 “그래, 그렇다면 마치 읽어볼까”라고 생각해주신다면 다행입니다.

그런데 요즘은 딥러닝 관련 라이브러리가 많이 공개되어, 누구나 손쉽게 이용할 수 있게 되었습니다. 사실 그런 라이브러리를 이용하면 딥러닝 프로그램을 만드는 게 어려운 일이 아닙니다. 그렇다면 왜 일부러 시간을 들여 처음부터 만들어보는 걸까요? 그 주된 이유는 무언가를 만드는 과정에서 배우는 게 많기 때문입니다.

무언가를 만드는 과정에서는 여러 가지 실험을 합니다. 어떤 때는 머리를 쥐어뜯으며 “왜 이렇게 되지?”라며 고민하게 됩니다. 이처럼 오랜 시간 고민하는 행위는 그 기술을 깊이 이해하는 데 소중한 지식이 됩니다. 그렇게 치분히 시간을 들여 얻은 지식은 기존 라이브러리를 활용하는 데도, 첨단의 논문을 읽는 데도, 순수한 나만의 시스템을 만드는 데도 반드시 도움이 될 것입니다. 그리고 무엇보다 직접 만든다는 것 자체가 즐거운 일 아닌가요? (신난다는데 다른 이유가 더 필요한가요?)

자, 준비는 갖춰졌습니다. 이제 딥러닝을 만드는 여행을 떠나봅시다!

## 감사의 말

우선 (기)계학습과 컴퓨터 과학( 등) 딥러닝 관련 기술 연구를 밀어붙인 연구자와 기술자께 감사합니다. 이 책을 쓸 수 있던 것도 그분들 덕분입니다. 또 서적과 웹 등으로 유용한 정보를 공개하는 분들에게 감사드립니다. 그중에서도 유용한 기술과 정보를 아낌없이 제공해준 스탠퍼드 대학교의 CS231n 공개강좌에서 많은 것을 느끼고 배웠습니다.

이 책이 나오기까지 많은 분이 도와주셨습니다. 팀 연구소의 카토오 테츠로, 기타 신야, 비영 유카, 나카노 코오타, 나카무라 쇼오 타츠, 임 테루 히로, 야마모토 료, 톰 스튜디오의 무토 젠지, 마스코 메구미, Flickrfit의 노무라 켄지, 텍사스 오스틴 대학교 JSPS 해외 특별 연구원의 단노 히데 터카. 이상의 분들은 이 책의 원고를 읽고 많은 조언을 주셨습니다. 모두 감사드립니다. 그럼에도 아직 부족하거나 잘못된 부분이 있다면 모두 저의 책임입니다.

마지막으로 이 책의 구성에서 완성까지 무려 1년 반동안 변함없이 응원해준 오라일리재팬의 미야 가와 나오카께 감사드립니다.

2016년 9월 어느 날  
시이토 고카

## CONTENTS

지은이 · 옮긴이 소개	4
총천사	5
옮긴이의 말	7
들어가며	9
감사의 말	14

## CHAPTER 1 헬로 파이썬

<b>1.1 파이썬이란?</b>	25
<b>1.2 파이썬 설치하기</b>	26
1.2.1 파이썬 버전	26
1.2.2 사용하는 외부 라이브러리	26
1.2.3 아나콘다 배포판	27
<b>1.3 파이썬 인터프리터</b>	27
1.3.1 선술 연산	28
1.3.2 자료형	29
1.3.3 변수	29
1.3.4 리스트	30
1.3.5 딕셔너리	31
1.3.6 bool	31
1.3.7 if문	32
1.3.8 for문	33
1.3.9 함수	33
<b>1.4 파이썬 스크립트 파일</b>	34
1.4.1 파일로 저장하기	34
1.4.2 클래스	34

## CONTENTS

---

<b>1.5 넘파이</b>	36
1.5.1 넘파이 가져오기	36
1.5.2 넘파이 배열 생성하기	37
1.5.3 넘파이의 선술 연산	37
1.5.4 넘파이의 N차원 배열	38
1.5.5 브로드캐스트	39
1.5.6 원소 접근	40
<b>1.6 matplotlib</b>	41
1.6.1 단순한 그래프 그리기	42
1.6.2 pyplot의 기능	43
1.6.3 이미지 표시하기	44
<b>1.7 정리</b>	45

---

## CHAPTER 2 퍼셉트론

---

<b>2.1 퍼셉트론이란?</b>	47
<b>2.2 단순한 논리 회로</b>	49
2.2.1 AND 게이트	49
2.2.2 NAND 게이트와 OR 게이트	49
<b>2.3 퍼셉트론 구현하기</b>	51
2.3.1 간단한 구현부터	51
2.3.2 기종치와 편향 도입	52
2.3.3 기종치와 편향 구현하기	52
<b>2.4 퍼셉트론의 학습</b>	54
2.4.1 도전! XOR 게이트	54
2.4.2 선형과 비선형	56

## CHAPTER 3 신경망

<b>2.5</b> 디층 퍼센트론이 출동한다면 .....	57
2.5.1 기존 계이트 조합하기 .....	57
2.5.2 XOR 계이트 구현하기   .....	59
<b>2.6</b> NAND에서 컴퓨터끼지 .....	61
<b>2.7</b> 정리 .....	62
<b>3.1</b> 퍼센트론에서 신경망으로 .....	63
3.1.1 신경망의 예 .....	64
3.1.2 퍼센트론 복습 .....	65
3.1.3 활성화 함수의 등장 .....	66
<b>3.2</b> 활성화 함수 .....	68
3.2.1 시그모이드 함수 .....	68
3.2.2 계단 함수 구현하기   .....	69
3.2.3 계단 함수의 그래프 .....	70
3.2.4 시그모이드 함수 구현하기   .....	72
3.2.5 시그모이드 함수와 계단 함수 비교 .....	74
3.2.6 비선형 함수 .....	75
3.2.7 ReLU 함수 .....	76
<b>3.3</b> 디차원 배열의 계산 .....	77
3.3.1 디차원 배열 .....	77
3.3.2 행렬의 곱 .....	79
3.3.3 신경망에서의 행렬 곱 .....	82
<b>3.4</b> 3층 신경망 구현하기   .....	83
3.4.1 표기법 설명 .....	83
3.4.2 각 층의 신호 전달 구현하기   .....	84
3.4.3 구현 정리 .....	89

## CONTENTS

---

<b>3.5 출력층 설계하기</b>	90
3.5.1 헝동 험수와 소프트맥스 험수 구현하기	91
3.5.2 소프트맥스 험수 구현 시 주의점	93
3.5.3 소프트맥스 험수의 특징	94
3.5.4 출력층의 뉴런 수 정하기	95
<b>3.6 손글씨 숫자 인식</b>	96
3.6.1 MNIST 데이터셋	96
3.6.2 신경망의 추론 처리	100
3.6.3 배치 처리	102
<b>3.7 정리</b>	105

## CHAPTER 4 신경망 학습

---

<b>4.1 데이터에서 학습한다!</b>	107
4.1.1 데이터 주도 학습	108
4.1.2 훈련 데이터와 시험 데이터	110
<b>4.2 손실 험수</b>	111
4.2.1 오차제곱합	112
4.2.2 교차 엔트로피 오차	113
4.2.3 미니배치 학습	115
4.2.4 (배치용) 교차 엔트로피 오차 구현하기	118
4.2.5 왜 손실 험수를 설정하는가?	119
<b>4.3 수치 미분</b>	121
4.3.1 미분	121
4.3.2 수치 미분의 예	124
4.3.3 편미분	125

<b>4.4 기울기</b>	127
4.4.1 경사법(경사 히경법)	129
4.4.2 신경망에서의 기울기	133
<b>4.5 학습 알고리즘 구현하기</b>	136
4.5.1 2층 신경망 클래스 구현하기	137
4.5.2 미니배치 학습 구현하기	141
4.5.3 사용 테이터로 평가하기	143
<b>4.6 정리</b>	146

## CHAPTER 5 오차역전파법

<b>5.1 계산 그래프</b>	148
5.1.1 계산 그래프로 풀다	148
5.1.2 규소적 계산	150
5.1.3 왜 계산 그래프로 푸는가?	151
<b>5.2 연쇄법칙</b>	152
5.2.1 계산 그래프의 역전파	153
5.2.2 연쇄법칙이란?	153
5.2.3 연쇄법칙과 계산 그래프	154
<b>5.3 역전파</b>	155
5.3.1 득셈 노드의 역전파	156
5.3.2 곱셈 노드의 역전파	157
5.3.3 시그모이드의 예	159
<b>5.4 단순한 계층 구현하기</b>	160
5.4.1 곱셈 계층	160
5.4.2 득셈 계층	162

## CONTENTS

---

<b>5.5 훈성화 함수 계층 구현하기</b>	165
5.5.1 ReLU 계층	165
5.5.2 Sigmoid 계층	167
<b>5.6 Affine/Softmax 계층 구현하기</b>	170
5.6.1 Affine 계층	170
5.6.2 배치용 Affine 계층	174
5.6.3 Softmax-with-Loss 계층	176
<b>5.7 오차역전파법 구현하기</b>	179
5.7.1 신경망 학습의 전체 그림	180
5.7.2 오차역전파법을 적용한 신경망 구현하기	180
5.7.3 오차역전파법으로 구현 기울기 검증하기	184
5.7.4 오차역전파법을 사용한 학습 구현하기	186
<b>5.8 정리</b>	187
<b>CHAPTER 6 학습 관련 기술들</b>	
<b>6.1</b>	
매개변수 경신	189
6.1.1 모험가 이이야기	190
6.1.2 획률적 경사 하강법(SGD)	190
6.1.3 SGD의 단점	192
6.1.4 모멘텀	194
6.1.5 AdaGrad	196
6.1.6 Adam	199
6.1.7 어느 경신 방법을 이용할 것인가?	200
6.1.8 MNIST 데이터셋으로 본 경신 방법 비교	201
<b>6.2</b>	
기종치의 조건!	202
6.2.1 초기값을 0으로 하면?	202

6.2.2 은닉층의 활성화값 분포	203
6.2.3 ReLU를 사용할 때의 기종치 조건값	207
6.2.4 MNIST 데이터셋으로 본 기종치 조건값 비교	209
<b>6.3 배치 정규화</b>	<b>210</b>
6.3.1 배치 정규화 알고리즘	210
6.3.2 배치 정규화의 효과	212
<b>6.4 다른 학습률을 위해</b>	<b>215</b>
6.4.1 오버피팅	215
6.4.2 기종치 감소	217
6.4.3 드롭아웃	219
<b>6.5 적절한 하이퍼파라미터 값 찾기</b>	<b>221</b>
6.5.1 검증 데이터	221
6.5.2 하이퍼파라미터 최적화	223
6.5.3 하이퍼파라미터 최적화 구현하기	224
<b>6.6 정리</b>	<b>226</b>
<hr/>	
<b>CHAPTER 7 합성곱 신경망(CNN)</b>	
<b>7.1 전체 구조</b>	<b>227</b>
<b>7.2 합성곱 계층</b>	<b>229</b>
7.2.1 윤전연결 계층의 문제점	229
7.2.2 합성곱 연산	230
7.2.3 패딩	232
7.2.4 스트라이드	233
7.2.5 3차원 데이터의 합성곱 연산	235
7.2.6 볼륨으로 생각하기	237
7.2.7 배치 처리	239

## CONTENTS

---

<b>7.3</b>	<b>풀링 계층</b>	<b>240</b>
7.3.1	풀링 계층의 특징	240
<b>7.4</b>	<b>합성곱/풀링 계층 구현하기</b>	<b>242</b>
7.4.1	4차원 배열	242
7.4.2	im2col로 텐서터 전개하기	243
7.4.3	합성곱 계층 구현하기	245
7.4.4	풀링 계층 구현하기	247
<b>7.5</b>	<b>CNN 구현하기</b>	<b>250</b>
<b>7.6</b>	<b>CNN 시각화하기</b>	<b>254</b>
7.6.1	1번째 층의 기종치 시각화하기	254
7.6.2	층 깊이에 따른 추출 정보 변화	256
<b>7.7</b>	<b>대표적인 CNN</b>	<b>257</b>
7.7.1	LeNet	257
7.7.2	AlexNet	258
<b>7.8</b>	<b>정리</b>	<b>259</b>
 <b>CHAPTER 8 딥러닝</b> <hr/>		
<b>8.1</b>	<b>더 깊게</b>	<b>261</b>
8.1.1	더 깊은 신경망으로	261
8.1.2	정확도를 더 높이려면	264
8.1.3	깊게 하는 이유	265
<b>8.2</b>	<b>딥러닝의 초기 역사</b>	<b>268</b>
8.2.1	이미지넷	268
8.2.2	VGG	270
8.2.3	GoogLeNet	271
8.2.4	ResNet	272

<b>8.3</b> 더 빠르게(딥러닝 고속화)	273
8.3.1 풀어야 할 숙제	274
8.3.2 GPU를 활용한 고속화	274
8.3.3 분산 학습	276
8.3.4 연산 정밀도와 비트 줄이기	277
<b>8.4</b> 딥러닝의 활용	278
8.4.1 시뮬 검출	278
8.4.2 분할	280
8.4.3 시즌 캡션 생성	281
<b>8.5</b> 딥러닝의 미래	283
8.5.1 이미지 스티일(회동) 변환	283
8.5.2 이미지 생성	284
8.5.3 자율 주행	286
8.5.4 Deep Q-Network(강화학습)	287
<b>8.6</b> 정리	289
<b>APPENDIX A Softmax-with-Loss 계층의 계산 그래프</b>	
<b>A.1</b> 순전파	292
<b>A.2</b> 역전파	294
<b>A.3</b> 정리	299
참고문헌	300
찾아보기	308



# 헬로 파이썬

파이썬이라는 프로그래밍 언어가 세상에 등장한 지도 이미 20년이 훌쩍 넘었습니다. 그 사이 파이썬은 독자적인 진화를 이루며 많은 개발자를 끌어들였습니다. 그리고 현재는 가장 인기 있고 많은 사람이 애용하는 프로그래밍 언어가 되었습니다.

앞으로 파이썬을 사용하여 딥러닝 시스템을 구현할 것입니다. 그에 앞서 이번 장에서는 파이썬에 대해서 간략히 소개하면서 그 사용법을 이해보겠습니다. 파이썬과 넘파이(Numpy, matplotlib)을 잘 아는 분은 이번 장을 건너뛰어도 상관없습니다.

## 1.1 파이썬이란?

파이썬은 간단하고 베우기 쉬운 프로그래밍 언어입니다. 오픈 소스라 무료로 자유롭게 이용할 수도 있고, 영어와 유사한 문법으로 프로그램을 작성할 수 있고 불편한 침과일 과정도 없어서 편리합니다. 그래서 프로그래밍 입문자에게 최적인 언어입니다. 실제로도 많은 대학교의 컴퓨터 과학 관련 학과에서 처음 가르치는 언어로 파이썬을 채용하는 사례가 아주 많습니다.

또, 파이썬 코드는 읽기 쉽고 성능도 뛰어납니다. 데이터가 많거나 빠른 응답이 필요할 때도 파이썬은 제 몫을 톡톡히 해냅니다. 그래서 초보자뿐 아니라 전문가들도 애용하죠. 실제로 구글과 마이크로소프트, 페이스북 등 IT의 첨단에서 겨루는 기업들은 파이썬을 자주 사용하고 있답니다.

파이썬은 과학 분야, 특히 기계학습과 데이터 과학 분야에서 널리 쓰입니다. 파이썬 자체의 뛰어난 성능에 넘파이와 사이파이[SciPy] 같은 수치 계산과 통계 처리를 다루는 턱월한 라이브러리가 더해져 데이터 과학 분야에서 확고한 위치를 차지하고 있죠. 나이가 딥러닝 프레임워크 쪽에서도 파이썬을 활용합니다. 예를 들어 카페[caffe], 텐서플로[Tensorflow], 체이너[Chainer], 테이노[Theano] 같은 유명 딥러닝 프레임워크들이 파이썬용 API를 제공합니다. 그래서 파이썬을 배우면 딥러닝 프레임워크를 사용할 때도 반드시 도움이 될 것입니다.

이처럼 파이썬은 특히 데이터 과학 분야에 아주 적합한 프로그래밍 언어입니다. 그래서 이 책의 목표인 딥러닝을 딥러닝을 딥러닝부터 만들기'를 달성하기 위한 언어로 선택된 것입니다.

## 1.2 파이썬 설치하기

그럼 파이썬을 PC에 설치합시다. 이번 절은 설치할 때 주의할 점을 설명합니다.

### 1.2.1 파이썬 버전

현재의 파이썬은 2.x와 3.x라는 두 가지 버전이 공존합니다. 3 버전이 나왔지만 아직 2 버전도 많이 이용되고 있죠. 그래서 파이썬을 처음 배울 때는 어느 버전을 설치할지 신중하게 선택해야 합니다. 이 둘은 100% 호환되는 게 아니니까요(정확히 말하면 '하위 호환성'이 없습니다). 즉, 파이썬 3로 짠 프로그램을 파이썬 2에서는 실행하지 못하는 일이 일어납니다. 이 책에서는 파이썬 3를 사용합니다. 만약 파이썬 2만 설치했다면, 파이썬 3도 추가로 설치하시기 바랍니다.

### 1.2.2 사용하는 외부 라이브러리

누차 강조했듯이 이 책의 목표는 딥러닝을 밀바닥부터 구현하는 것입니다. 그래서 외부 라이브러리는 최소한만 사용한다는 것이 기본 방침이지만, 다음의 두 라이브러리는 예외로 하겠습니다. 하나는 넘파이, 다른 하나는 matplotlib입니다. 이 라이브러리를 사용하는 이유는 효율적으로 딥러닝을 구현하기 위해서입니다.

넘파이는 수치 계산용 라이브러리입니다. 넘파이에는 고도의 수학 알고리즘과 베열(행렬)을 조작하기 위한 편리한 메서드가 많이 준비되어 있습니다. 이 메서드들을 이용하면 딥러닝을 훨씬 효율적으로 구현할 수 있습니다.

matplotlib은 그레프를 그려주는 라이브러리입니다. matplotlib을 이용하면 실험 결과를 시각화하거나 딥러닝 실행 과정의 중간 데이터를 시각적으로, 즉 눈으로 확인할 수 있습니다. 이 책에서는 이러한 라이브러리를 사용하여 딥러닝을 구현합니다.

**NOTE** 이 책에서는 다음의 프로그래밍 언어와 라이브러리를 사용합니다.

- 파이썬 3
- 넘파이
- matplotlib

다음으로 파이썬 설치 방법을 설명합니다. 이미 설치한 분은 건너뛰시면 됩니다.

### 1.2.3 아나콘다 배포판

파이썬을 설치하는 방법은 다양하지만, 이 책에서는 아나콘다Anaconda라는 배포판을 이용하기를 권합니다. 배포판이란 사용자가 설치를 한 번에 수행할 수 있도록 필요한 라이브러리 등을 하나로 정리해둔 것입니다. 그중 아나콘다는 데이터 분석에 중점을 둔 배포판입니다. 방금 설명한 넘파이와 matplotlib을 포함해 데이터 분석에 유용한 라이브러리가 포함되어 있습니다.

앞서 말한 것처럼 이 책에서는 파이썬 3를 사용합니다. 그래서 아나콘다 배포판도 3 버전용을 설치합니다. 그럼, 다음 주소에서 자신의 OS에 맞는 배포판을 내려받아 설치하세요.

- <https://www.anaconda.com/distribution>

## 1.3 파이썬 인터프리터

파이썬을 설치했다면 파이썬 버전을 먼저 확인합니다. 터미널(윈도우라면 명령 프롬프트)에서 python --version 명령어를 실행해보세요. 이 명령은 설치된 파이썬의 버전을 출력합니다.

```
$ python --version
Python 3.5.2 :: Anaconda 4.2.0 (x86_64)
```

이처럼 “Python 3.x.x”라 표시되면 파이썬 3가 제대로 설치된 것입니다(설치한 버전에 따라 숫자는 달라지겠죠?). 이어서 python이라고 입력하여 파이썬 인터프리터를 시작해보세요.

```
$ python
Python 3.5.2 |Anaconda 4.2.0 (x86_64)| (default, Jul 2 2016, 17:52:12)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

파이썬 인터프리터는 ‘내화 모드’라 하여, 개발자와 파이썬이 대화하듯 프로그래밍할 수 있습니다. ‘내화하듯’이라는 것은, 예를 들면 개발자가 “1+2는?”이라고 물으면 파이썬 인터프리터가 글바로 “3입니다”라고 대답한다는 의미입니다. 그럼 실제로 입력해보겠습니다.

```
>>> 1 + 2
3
```

이처럼 파이썬 인터프리터에서는 대화식으로 프로그래밍할 수 있습니다. 이번 장에서는 이 대화 모드를 사용하여 파이썬 프로그래밍을 간단히 실습해보려 합니다.

### 1.3.1 산술 연산

덧셈과 곱셈 등의 산술 연산은 다음과 같이 할 수 있습니다.

```
>>> 1 - 2
-1
>>> 4 * 5
20
>>> 7 / 5
1.4
>>> 3 ** 2
9
```

\*은 곱셈, /는 나눗셈, \*\*는 거듭제곱을 의미합니다( $3 ** 2$ 는 3의 2제곱). 참고로, 파이썬 2에서는 정수끼리 계산한 결과는 정수입니다. 예를 들어  $7 \div 5$ 의 결과는 1입니다. 한편, 파이썬 3에서는 정수를 나눈 결과는 실수(부동소수점)가 됩니다.

### 1.3.2 자료형

프로그래밍 언어에는 자료형 data type이라는 것이 있습니다. 자료형이란 데이터의 성질을 나타내는 것으로, 예를 들어 정수, 실수, 문자열과 같은 형태가 있습니다. 파이썬에는 type() 함수로 특정 데이터의 자료형을 알아볼 수 있습니다.

```
>>> type(10)
<class 'int'>
>>> type(2.718)
<class 'float'>
>>> type("hello")
<class 'str'>
```

즉, 10은 int(정수), 2.718은 float(실수), "hello"는 str(문자열)형임을 알 수 있습니다. 또한 자료형과 클래스class라는 말을 같은 의미로 사용하는 경우가 있습니다. 방금 예제에서의 <class "int">은 "10은 int라는 클래스(자료형)다"로 해석하면 됩니다.

### 1.3.3 변수

x와 y 등의 알파벳을 사용하여 변수variable를 정의할 수 있습니다. 또한, 변수를 사용하여 계산하거나 변수에 다른 값을 대입할 수도 있습니다.

```
>>> x = 10          # 초기화
>>> print(x)      # x의 값 출력
10
>>> x = 100         # 변수에 값 대입
>>> print(x)
100
>>> y = 3.14
>>> x * y
```

```
314.0
>>> type(x * y)
<class 'float'>
```

파이썬은 **통적 언어**로 분류되는 프로그래밍 언어입니다. 동적이라 함은 변수의 자료형을 상황에 맞게 자동으로 결정한다는 뜻입니다. 앞의 예에서 x의 자료형이 int(정수)라는 것을 사용자가 명시한 적이 없죠? 하지만 10이라는 정수로 초기화할 때, x의 형태가 int임을 파이썬이 스스로 판단하는 것입니다. 또, 정수와 실수를 곱한 결과는 실수가 되었습니다(자동 형변환). 마지막으로 #은 주석의 시작을 알리는 문자입니다. # 이후의 문자는 모두 무시해버립니다.

## 1.3.4 리스트

여러 테이터를 리스트(list)로도 정리할 수 있습니다.

```
>>> a = [1, 2, 3, 4, 5] # 리스트 생성
>>> print(a) # 리스트의 내용 출력
[1, 2, 3, 4, 5]
>>> len(a) # 리스트의 길이 출력
5
>>> a[0] # 첫 원소에 접근
1
>>> a[4] # 다섯 번째 원소에 접근
5
>>> a[4] = 99 # 값 대입
>>> print(a)
[1, 2, 3, 4, 99]
```

원소에 접근할 때는 a[0]처럼 합니다. [] 안의 수를 인덱스(색인)라 하며 인덱스는 0부터 시작합니다(인덱스 0이 첫 번째 원소를 가리킵니다). 또 파이썬 리스트에는 **슬라이싱(slicing)**이라는 편리한 기법이 준비되어 있습니다. 슬라이싱을 이용하면 범위를 지정해 원하는 부분 리스트를 얻을 수 있습니다.

```
>>> print(a)
[1, 2, 3, 4, 99]
>>> a[0:2] # 인덱스 0부터 2까지 열기(2번째는 포함하지 않는다!)
[1, 2]
```

```
>>> a[1:] # 인덱스 1부터 끝까지 얻기
[2, 3, 4, 99]
>>> a[:3] # 처음부터 인덱스 3까지 얻기(3번째는 포함하지 않는다!)
[1, 2, 3]
>>> a[:-1] # 처음부터 마지막 원소의 1개 앞까지 얻기
[1, 2, 3, 4]
>>> a[:-2] # 처음부터 마지막 원소의 2개 앞까지 얻기
[1, 2, 3]
```

리스트를 슬라이싱하면 `a[0:2]`처럼 씁니다. `a[0:2]`는 인덱스 0부터 1(2보다 하나 앞까지)의 원소를 까냅니다. 인덱스 번호 -1은 마지막 원소, -2는 끝에서 한 개 앞의 원소에 해당합니다.

### 1.3.5 딕셔너리

리스트는 인덱스 번호로 0, 1, 2, ... 순으로 값을 저장하는 반면, 딕셔너리(dictionary)는 `key`와 `value`을 한 쌍으로 저장합니다. 즉, 영한사전처럼 단어와 그 의미를 짹지어 저장합니다.

```
>>> me = {'height':180} # 딕셔너리 생성
>>> me['height'] # 원소에 접근
180
>>> me['weight'] = 70 # 새 원소 추가
>>> print(me)
{'height': 180, 'weight': 70}
```

### 1.3.6 bool

파이썬에는 `bool` 혹은 `布尔型`이라는 자료형이 있습니다. 이 자료형은 `True`(참)와 `False`(거짓)라는 두 값 중 하나를 취합니다. 또 `bool`에는 `and`, `or`, `not` 연산자를 사용할 수 있습니다(수치용 연산자로는 `+`, `-`, `*`, `/` 등이 있듯이) 자료형에 따라 사용할 수 있는 연산자가 정해져 있습니다).

```
>>> hungry = True # 배가 고프다.
>>> sleepy = False # 출근하지 않다.
```

```
>>> type(hungry)
<class 'bool'>
>>> not hungry
False
>>> hungry and sleepy # 배가 고프다 그리고 졸리지 않다.
False
>>> hungry or sleepy # 배가 고프다 또는 졸리지 않다.
True
```

## 1.3.7 if 문

조건에 따라서 달리 처리하려면 **if/else** 문을 사용합니다.

```
>>> hungry = True
>>> if hungry:
...     print("I'm hungry")
...
I'm hungry
>>> hungry = False
>>> if hungry:
...     print("I'm hungry")      # 들여쓰기는 공백 문자로
... else:                         # 줄여쓰기와 같은 경우
...     print("I'm not hungry")
...     print("I'm sleepy")
...
I'm not hungry
I'm sleepy
```

파이썬에서는 공백 문자가 중요한 의미를 지닙니다. 이번 if 문에서도 if hungry: 다음 줄은 앞쪽에 4개의 공백 문자가 있습니다. 이 들여쓰기는 지난 조건(if hungry)이 충족될 때 실행되며 코드를 표현합니다.

**WARNING** 공백 대신 탭 tab 문자를 써도 되지만 파이썬에서는 공백 문자 쪽을 권장합니다. 그리고 한 단계 더 들여 쓸 때마다 공백 4개씩을 더 추가하는 것이 일반적입니다.

## 1.3.8 for 문

반복(루프) 처리에는 **for 문**을 사용합니다.

```
>>> for i in[1, 2, 3]:  
...     print(i)  
...  
1  
2  
3
```

여기에서는 [1, 2, 3]이라는 리스트 안의 원소를 하나씩 출력하는 예를 보여졌습니다. for ... in ... 구문을 사용하면 리스트 등 데이터 집합의 각 원소에 차례로 접근할 수 있습니다.

## 1.3.9 함수

특정 기능을 수행하는 일련의 명령들을 묶어 하나의 **함수** function로 정의할 수 있습니다.

```
>>> def hello():  
...     print("Hello World!")  
...  
>>> hello()  
Hello World!
```

함수는 인수를 취할 수 있습니다. 또한, + 연산자를 사용하여 문자열을 이어 붙일 수 있습니다.

```
>>> def hello(object):  
...     print("Hello " + object + "!")  
...  
>>> hello("cat")  
Hello cat!
```

파이썬 인터프리터를 종료하려면 리눅스와 맥에서는 **[Ctrl]+[D]** (**[Ctrl]** 키를 누른 상태에서 **[D]** 키를 누른다)를 입력합니다. 윈도우에서는 **[Ctrl]+[Z]** (**[Ctrl]** 키를 누르고 **[Enter]** 키를 누릅니다).

## 1.4 파이썬 스크립트 파일

지금까지 파이썬 인터프리터를 활용하는 예를 보았습니다. 파이썬 인터프리터는 파이썬 코드를 대화식으로 실행해보며 간단한 실현을 수행하기에 딱 좋습니다. 그러나 긴 작업을 수행해야 한다면, 매번 코드를 입력해야 하는 이 방식은 조금 불편하겠죠. 이럴 때는 파이썬 프로그램을 파일로 저장하고 그 파일을 (함께) 실행하는 방법이 있습니다. 이번 절에서는 그런 파이썬 스크립트 파일의 예를 살펴보겠습니다.

### 1.4.1 파일로 저장하기

텍스트 편집기를 열고 hungry.py라는 파일을 작성합니다. hungry.py는 다음의 한 줄만으로 구성된 파일입니다.

```
print("I'm hungry!")
```

이어서 터미널을 열고 앞의 hungry.py를 저장한 디렉터리로 이동합니다. 그런 다음 파일 이름인 “hungry.py”를 인수로 python 명령을 실행합니다. 여기에서는 hungry.py가 ~/deep-learning-from-scratch/ch01 디렉터리에 있다고 가정합니다.

```
$ cd ~/deep-learning-from-scratch/ch01 # 디렉터리로 이동  
$ python hungry.py  
I'm hungry!
```

이처럼 파이썬 코드를 담은 파일을 인수로 지정해 파이썬 프로그램을 실행할 수 있습니다.

### 1.4.2 클래스

지금까지 int와 str 등의 자료형을 살펴봤습니다(그리고 type() 함수로는 원하는 테이터의 자료형을 알아낼 수 있었죠). 이들은 내장된 자료형, 즉 파이썬이 기본으로 제공하는 자료형입니다. 이번 절에서는 새로운 클래스를 정의합니다. 개발자가 직접 클래스를 정의하면 독자적인 자료형을 만들 수 있습니다. 또한, 클래스에는 그 클래스만의 전용 함수(메서드)와 속성을 정의

할 수도 있습니다.

파이썬에서는 class라는 키워드를 사용하여 클래스를 정의합니다. 클래스의 구조는 다음과 같습니다.

```
class 클래스 이름:  
    def __init__(self, 인수, ...):      # 생성자  
        ...  
        def 메서드 이름 1(self, 인수, ...):  # 메서드 1  
            ...  
        def 메서드 이름 2(self, 인수, ...):  # 메서드 2  
            ...
```

클래스 정의에는 `_init_`라는 특별한 메서드가 있는데, 클래스를 초기화하는 방법을 정의합니다. 이 초기화용 메서드를 **생성자** constructor라고도 하며, 클래스의 인스턴스가 만들어질 때 한 번만 불립니다. 또, 파이썬에서는 메서드의 첫 번째 인수로 자신(자신의 인스턴스)을 나타내는 `self`를 명시적으로 쓰는 것이 특징입니다 (다른 언어를 쓴던 사람은 이처럼 `self`를 쓰는 규칙을 기묘하게 느낄지도 모르겠네요).

그럼 간단한 클래스를 하나 만들어보겠습니다. 다음 코드를 `man.py` 파일로 저장하세요.

```
class Man:  
    def __init__(self, name):  
        self.name = name  
        print("Initialized!")  
  
    def hello(self):  
        print("Hello " + self.name + "!")  
  
    def goodbye(self):  
        print("Good-bye " + self.name + "!")  
  
m = Man("David")  
m.hello()  
m.goodbye()
```

○ 제 터미널에서 `man.py`를 실행합니다.

```
$ python man.py
Initialized!
Hello David!
Good-bye David!
```

여기에서는 Man이라는 새로운 클래스를 정의했습니다. 그리고 Man 클래스에서 m이라는 인스턴스(객체)를 생성합니다.

Man의 생성자(초기화 메서드)는 name이라는 인수를 받고, 그 인수로 인스턴스 변수인 self.name을 초기화합니다. 인스턴스 변수는 인스턴스별로 저장됩니다. 파이썬에서 self.name처럼 self 다음에 속성 이름을 써서 인스턴스 변수를 작성하거나 접근할 수 있습니다.

## 1.5 넘파이

딥러닝을 구현하다 보면 배열이나 행렬 계산이 많이 등장합니다. 넘파이의 배열 클래스인 numpy.array에는 편리한 메서드가 많이 준비되어 있어, 딥러닝을 구현할 때 이 메서드들을 이용합니다. 이번 절에서는 앞으로 사용할 넘파이에 대해서 간략히 설명합니다.

### 1.5.1 넘파이 가져오기

넘파이는 외부 라이브러리를 임기 위해서 import 문을 이용합니다. 여기에서는 import numpy as np라고 썼는데, 직역하면 “numpy를 np라는 이름으로 가져와라”가 됩니다. 이렇게 해두면 앞으로 넘파이가 제공하는 메서드를 np를 통해 참조할 수 있습니다.

```
>>> import numpy as np
```

파이썬에서는 라이브러리를 임기 위해서 import 문을 이용합니다. 여기에서는 import numpy as np라고 썼는데, 직역하면 “numpy를 np라는 이름으로 가져와라”가 됩니다. 이렇게 해두면 앞으로 넘파이가 제공하는 메서드를 np를 통해 참조할 수 있습니다.

## 1.5.2 넘파이 배열 생성하기

넘파이 배열을 만들 때는 np.array() 메서드를 이용합니다. np.array()는 파이썬의 리스트를 인수로 받아 넘파이 라이브러리가 제공하는 특수한 형태의 배열(numpy.ndarray)을 반환합니다.

```
>>> x = np.array([1.0, 2.0, 3.0])
>>> print(x)
[1. 2. 3.]
>>> type(x)
<class 'numpy.ndarray'>
```

## 1.5.3 넘파이의 산술 연산

다음은 넘파이 배열로 산술 연산을 수행하는 예입니다.

```
>>> x = np.array([1.0, 2.0, 3.0])
>>> y = np.array([2.0, 4.0, 6.0])
>>> x + y # 원소별 덧셈
array([ 3.,  6.,  9.])
>>> x - y
array([-1., -2., -3.])
>>> x * y # 원소별 곱셈
array([ 2.,  8., 18.])
>>> x / y
array([ 0.5,  0.5,  0.5])
```

여기에서 주의할 점은 배열 x와 y의 원소 수가 같다는 것입니다(둘 다 원소를 3개씩 갖는 1차원 배열). x와 y의 원소 수가 같다면 산술 연산은 각 원소에 대해서 행해집니다. 원소 수가 다르면 오류가 발생하니 원소 수 맞추기는 중요하답니다. 참고로, ‘원소별’이라는 말은 영어로 element-wise라고 합니다. 예컨대 ‘원소별 곱셈’은 element-wise product라고 합니다.

넘파이 배열은 원소별 계산뿐 아니라 넘파이 배열과 수치 하나(스칼라값)의 조합으로 된 산술 연산도 수행할 수 있습니다. 이 경우 스칼라값과의 계산이 넘파이 배열의 원소별로 한 번씩 수행됩니다. 이 기능을 **브로드캐스트**라고 합니다(자세한 것은 1.5.5절에서 설명하겠습니다).

```
>>> x = np.array([1.0, 2.0, 3.0])
>>> x / 2.0
array([ 0.5,  1.,  1.5])
```

## 1.5.4 넘파이의 N차원 배열

넘파이는 1차원 배열(1줄로 늘어선 배열)뿐 아니라 다차원 배열도 작성할 수 있습니다. 예를 들어 2차원 배열(행렬)은 다음과처럼 작성합니다.

```
>>> A = np.array([[1, 2], [3, 4]])
>>> print(A)
[[1 2]
 [3 4]]
>>> A.shape
(2, 2)
>>> A.dtype
dtype('int64')
```

방금  $2 \times 2$ 의 A라는 행렬을 작성했습니다. 행렬의 형상\*은 shape으로, 행렬에 담긴 원소의 자료형은 dtype으로 알 수 있습니다. 이어서 행렬의 산술 연산을 봅시다.

```
>>> B = np.array([[3, 0], [0, 6]])
>>> A + B
array([[ 4,  2],
       [ 3, 10]])
>>> A * B
array([[ 3,  0],
       [ 0, 24]])
```

형상\* 같은 행렬끼리면 행렬의 산술 연산도 대응하는 원소별로 계산됩니다. 배열과 마찬가지로 말이죠. 행렬과 스칼라값의 산술 연산도 가능합니다. 이때도 배열과 마찬가지로 브로드캐스팅 기능이 작동합니다.

\* 옮긴이 이 책에서는 행렬을 포함한 N차원 배열에서 그 배열의 각 차원의 크기(원소 수)를 배열의 '형상'이라 하겠습니다.

```
>>> print(A)
[[1 2]
 [3 4]]
>>> A * 10
array([[ 10,  20],
       [ 30,  40]])
```

**NOTE** 넘파이 배열(nparray)은 N차원 배열을 작성할 수 있습니다. 1차원 배열 2차원 배열 3차원 배열처럼 원하는 차수의 배열을 만들 수 있다는 뜻입니다. 수학에서는 1차원 배열은 벡터(vector), 2차원 배열은 행렬(matrix)이라고 부릅니다. 또 벡터와 행렬을 일반화한 것을 텐서(tensor)라 합니다.\*\* 0 차원에서는 기본적으로 2차원 배열을 ‘행렬’, 3차원 이상의 배열을 다차원 배열이라 하겠습니다.

## 1.5.5 브로드캐스트

넘파이에서는 형상이 다른 배열끼리도 계산할 수 있습니다. 앞의 예에서는  $2 \times 2$  행렬 A에 스칼라값 10을 곱했습니다. 이때 [그림 1-1]과 같이 10이라는 스칼라값이  $2 \times 2$  행렬로 확장된 후 연산이 이루어집니다. 이 편의적인 기능을 **브로드캐스트(broadcast)**라고 합니다.

그림 1-1 브로드캐스트의 예 : 스칼라값인 10이  $2 \times 2$  행렬로 확장된다.

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} * \boxed{10} = \begin{matrix} 10 & 20 \\ 30 & 40 \end{matrix}$$

다른 예를 살펴봅시다.

```
>>> A = np.array([[1, 2], [3, 4]])
>>> B = np.array([10, 20])
>>> A * B
array([[ 10,  40],
       [ 30,  80]])
```

여기에서는 [그림 1-2]처럼 1차원 배열인 B가 ‘똑똑하게도’ 2차원 배열 A와 똑같은 형상으로

\*\* 출처: 구글의 딥러닝 프레임워크인 텐서플로의 이름이 여기서 유래했습니다. 텐서(tensor)가 신경망을 타고 흐른다(flow)는 뜻이죠.

변형된 후 원소별 연산이 이루어집니다.

그림 1-2 브로드캐스트의 예 2

$$\begin{array}{c} \text{1 2} \\ \text{3 4} \end{array} * \begin{array}{c} \text{10 20} \\ \text{3 4} \end{array} = \begin{array}{c} \text{1 2} \\ \text{3 4} \end{array} * \begin{array}{c} \text{10 20} \\ \text{10 20} \end{array} = \begin{array}{c} \text{10 40} \\ \text{30 80} \end{array}$$

이처럼 넘파이가 제공하는 브로드캐스트 기능 덕분에 형상이 다른 배열끼리의 연산을 스마트하게 할 수 있습니다.

## 1.5.6 원소 접근

원소의 인덱스는 0부터 시작합니다(기억하시죠?). 그리고 각 원소에 접근하려면 다음과 같아 합니다.

```
>>> x = np.array([[51, 55], [14, 19], [0, 4]])
>>> print(x)
[[51 55]
 [14 19]
 [ 0  4]]
>>> x[0]          # 0행
array([51, 55])
>>> x[0][1]       # (0, 1) 위치의 원소
55
```

for 문으로도 각 원소에 접근할 수 있습니다.

```
>>> for row in x:
...     print(row)
...
[51 55]
[14 19]
[ 0  4]
```

넘파이는 지금까지의 방법 외에도, 인덱스를 배열로 지정해 한 번에 여러 원소에 접근할 수도 있습니다.

```
>>> X = X.flatten()          # X를 1차원 배열로 변환(평탄화)
>>> print(X)
[51 55 14 19  0  4]
>>> X[[np.array([0, 2, 4])]] # 인덱스가 [0, 2, 4인 원소 얻기
array([51, 14,  0])
```

이 기법을 응용하면 특정 조건을 만족하는 원소만 얻을 수 있습니다. 예컨대 다음과 같이 배열 X에서 15 이상인 값만 구할 수 있습니다.

```
>>> X > 15
array([ True,  True, False,  True, False, False])
>>> X[X>15]
array([51, 55, 19])
```

넘파이 배열에 부등호 연산자를 사용한(앞 예에서 X>15) 결과는 bool 배열입니다.\* 여기에서는 이 bool 배열을 사용해 배열 X에서 True에 해당하는 원소, 즉 값이 15보다 큰 원소만 꺼내고 있습니다.

**NOTE** 파이썬 같은 동적 언어는 C나 C++ 같은 정적 언어(컴파일 언어)보다 처리 속도가 높다고 합니다. 실제로 무거운 작업을 할 때는 C/C++로 작성한 프로그램을 쓰는 편이 좋습니다. 그래서 파이썬에서 빠른 성능이 요구될 경우 해당 부분을 C/C++로 구현하곤 합니다. 그때 파이썬은 C/C++로 쓰인 프로그램을 호출해주는, 이를바 ‘중개자’ 같은 역할을 합니다. 넘파이도 주된 처리는 C와 C++로 구현하습니다. 그래서 성능을 해치지 않으면서 파이썬의 편리한 문법을 사용할 수 있는 것이죠.

## 1.6 matplotlib

딥러닝 실험에서는 그래프 그리기와 테이터 시작화도 중요하답니다. **matplotlib**은 그래프를 그려주는 라이브러리입니다. matplotlib을 사용하면 그래프 그리기와 테이터 시작화가 쉬워집니다. 이번 절에서는 그래프를 그리고 이미지를 화면에 표시하는 방법을 설명합니다.

\* *옮긴이* 넘파이 배열에 부등호 연산을 수행하면 배열의 원소 각각에 부등호 연산을 수행한 bool 배열이 생성됩니다.

### 1.6.1 단순한 그래프 그리기

그래프를 그리려면 matplotlib의 **pyplot** 모듈을 이용합니다. 당장 sin 함수를 그리는 예를 살펴봅시다.

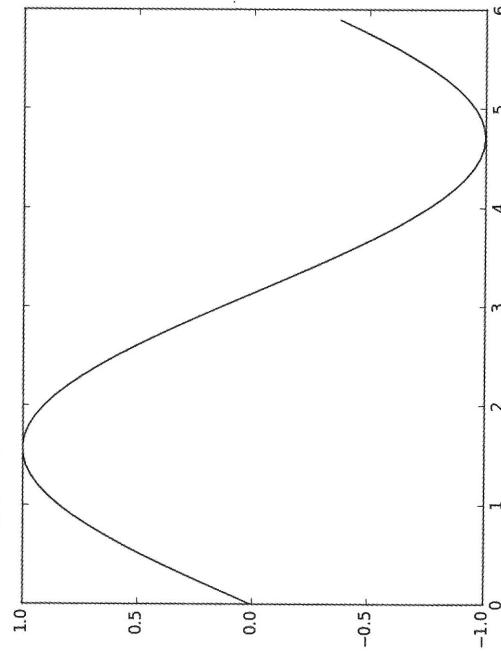
```
import numpy as np
import matplotlib.pyplot as plt

# 데이터 준비
x = np.arange(0, 6, 0.1) # 0에서 6까지 0.1 간격으로 생성
y = np.sin(x)

# 그래프 그리기
plt.plot(x, y)
plt.show()
```

이 코드에서는 넘파이의 `arange` 메서드로  $[0, 0.1, 0.2, \dots, 5.8, 5.9]$ 라는 데이터를 생성하여 변수 `x`에 할당했습니다. 그다음 줄에서는 `x`의 각 원소에 넘파이의 `sin` 함수인 `np.sin()`을 적용하여 변수 `y`에 할당합니다. 이제 `x`와 `y`를 인수로 `plt.plot` 메서드를 호출해 그래프를 그립니다. 마지막으로 `plt.show()`를 호출해 그레프를 화면에 출력하고 끝납니다. 이 코드를 실행하면 [그림 1-3]의 이미지가 그려집니다.

그림 1-3 sin 함수 그래프



## 1.6.2 pyplot의 기능

여기에 cos 함수도 추가로 그려보겠습니다. 또, 제목과 각 축의 이름(레이블) 표시 등, **pyplot**의 다른 기능도 사용해보겠습니다.

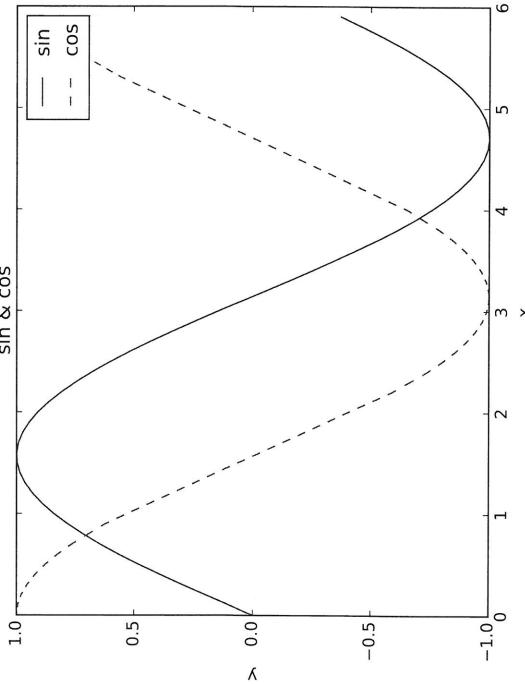
```
import numpy as np
import matplotlib.pyplot as plt

# 데이터 준비
x = np.arange(0, 6, 0.1) # 0에서 6까지 0.1 간격으로 생성
y1 = np.sin(x)
y2 = np.cos(x)

# 그레프 그리기
plt.plot(x, y1, label="sin")
plt.plot(x, y2, linestyle="--", label="cos") # cos 함수는 점선으로 그리기
plt.xlabel("x") # x축 이름
plt.ylabel("y") # y축 이름
plt.title('sin & cos') # 제목
plt.legend()
plt.show()
```

결과는 [그림 1-4]와 같습니다. 그레프의 제목과 축 이름이 보일 겁니다.

그림 1-4 sin 함수와 cos 함수 그레프



### 1.6.3 이미지 표시하기

matplotlib에는 이미지를 표시해주는 메서드인 `imshow()`도 준비되어 있습니다. 이미지를 읽어 들일 때는 `matplotlib.image` 모듈의 `imread()` 메서드를 이용합니다. 예를 보시죠.

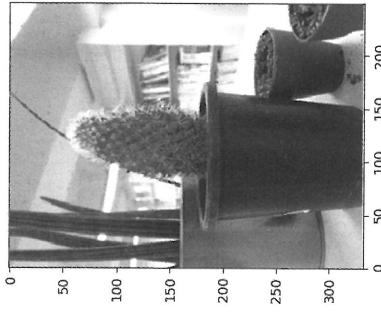
```
import matplotlib.pyplot as plt
from matplotlib.image import imread

img = imread('cactus.png') # 이미지 읽어오기(적절한 경로를 설정하세요!)

plt.imshow(img)
plt.show()
```

이 코드를 실행하면 [그림 1-5]처럼 읽어들인 이미지가 표시됩니다.

그림 1-5 이미지 표시하기



앞의 코드에서는 `cactus.png`라는 이미지 파일이 현재 작업 디렉터리에 있다고 가정했습니다. 여러분은 자신의 환경에 맞게 파일 이름과 경로를 적절히 수정해야 합니다. 이 책이 제공하는 소스 코드에서는 `dataset` 디렉터리에서 이 `cactus.png` 파일을 찾을 수 있습니다. 예를 들어 파이썬 인터프리터로 `ch01` 디렉터리에서 이 코드를 실행한다면 이미지 경로를 '`'cactus.png'`'에서 '`./dataset/cactus.png`'로 변경하면 올바르게 작동합니다.

## 1.7 정리

이번 장은 딥러닝으로의 본격적인 여정을 위한 준비 과정입니다. 특히, 딥러닝(신경망)을 구현하는 데 필요한 프로그래밍의 기본을 중심으로 살펴보았습니다. 다음 장에서 파이썬으로 실제로 작동하는 코드를 작성해보면서 딥러닝의 세계로 떠나볼겁니다.

이번 장에서는 파이썬에 대해 우리에게 필요한 최소한만 설명했습니다. 더 깊게 알고 싶은 분을 위해 좋은 책을 두 권 소개해드리죠. 먼저『처음 시작하는 파이썬』(한빛미디어, 2015)<sup>[1]</sup>입니다. 이 책은 파이썬 프로그래밍을 기초부터 응용까지 친절하게 설명해주는 실천적인 입문서입니다. 넘파이에 대해서는『파이썬 라이브러리를 활용한 데이터 분석』(한빛미디어, 2013)<sup>[2]</sup>을 추천합니다. 책 외에는 〈Scipy 강의 노트〉라는 웹 사이트<sup>[3]</sup>가 과학 기술에서의 계산을 주제로 넘파이와 matplotlib을 잘 설명하고 있으니 참고하시기 바랍니다.

### 이번 장에서 배운 내용

- 파이썬은 간단하고 익히기 쉬운 프로그래밍 언어다.
- 파이썬은 오픈 소스에서 자유롭게 사용할 수 있다.
- 이 책은 딥러닝 구현에 파이썬 3 버전을 이용한다.
- 외부 라이브러리는 넘파이와 matplotlib을 이용한다.
- 파이썬을 실행하는 방식에는 '인터프리터'와 '스크립트 파일' 두 가지가 있다.
- 파이썬에서는 함수와 클래스 같은 모듈로 구현을 정리할 수 있다.
- 넘파이는 다차원 배열을 다루는 편리한 메서드를 많이 제공한다.

