O'REILLY®

Глубокое обучение в биологии и медицине

Бхарат Рамсундар, Питер Истман,
Патрик Уолтерс, Виджай Панде

ДМК

Bharath Ramsundar, Peter Eastman, Patrick Walters, and Vijay Pande

Deep Learning for the Life Sciences

O'REILLY®
Beijing • Boston • Farnham • Sebastopol • Tokyo

Бхарат Рамсундар, Питер Истман, Патрик Уолтерс и Виджай Панде

Глубокое обучение в биологии и медицине

ДМК Пресс
издательство
Москва, 2020

УДК 004.891
ББК 32.972.13
Р21

Рамсундар Б., Истман П., Уолтерс П., Панде В.
Р21 Глубокое обучение в биологии и медицине / пер. с анг. В. С. Яценкова. – М.: ДМК Пресс, 2020. – 200 с.: ил.

ISBN 978-5-97060-791-6

Глубокое обучение добилось впечатляющих успехов во многих отраслях. Сейчас оно все глубже проникает в прикладные научные исследования, в частности биологию и смежные дисциплины. Эта книга рассказывает о применении глубокого обучения в геномике, химии, биофизике, микроскопии, медицине и других направлениях современных исследований всего, что связано с живыми организмами.

Издание будет полезно широкому кругу специалистов, связанных с анализом данных в химии, биологии и медицине, а также разработчикам ПО для них и студентам вузов.

УДК 004.891
ББК.32.972.13

ISBN 978-1-492-03983-9 (анг.)
ISBN 978-5-97060-791-6 (рус.)

Содержание

Предисловие 8

Об авторах 11

Глава 1. Глубокое обучение и науки о жизни 13
Почему все говорят о глубоком обучении? 13
Современные науки о жизни – это науки о данных 14
О чем эта книга? 15

Глава 2. Введение в глубокое обучение 19
Линейные модели 20
Многослойные персептроны 21
Обучение модели 24
Проверка модели 26
Регуляризация 27
Оптимизация гиперпараметров 28
Другие типы моделей 29
Сверточные нейронные сети 30
Рекуррентные нейронные сети 31
Дополнительное чтение 32

Глава 3. Машинное обучение с DeepChem 33
Наборы данных DeepChem 34
Обучение модели для предсказания токсичности молекул 35
Пример: обучение модели MNIST 42
Набор данных распознавания цифр MNIST 42
Сверточная архитектура для набора MNIST 43
Заключение 47

Глава 4. Машинное обучение и молекулы 48
Что такое молекула? 48
Что такое внутримолекулярные связи? 50
Ковалентные связи 51
Нековалентные связи 51
Молекулярные графы 52
Конформации молекулы 53
Хиральность молекул 54
Фичеризация молекулы 55
Строки SMILES и пакет RDKit 55
Расширенные отпечатки связей 56
Молекулярные дескрипторы 57
Графовые свертки 57

Обучение модели для прогнозирования растворимости ... 58
MoleculeNet ... 60
Строки SMARTS ... 60
Заключение ... 63

Глава 5. Глубокое обучение и биофизика ... 64

Белковые структуры ... 65
Белковые последовательности ... 67
Общие принципы связывания с белками ... 70
Биофизическая фичеризация ... 71
Координатная фичеризация ... 71
Атомная фичеризация ... 76
Пример использования PDBBind ... 76
PDBBind Dataset ... 76
Представление набора данных PDBBind ... 79
Заключение ... 82

Глава 6. Глубокое обучение и геномика ... 85

ДНК, РНК и белки ... 85
Реальное положение дел ... 87
Сайты связывания и факторы транскрипции ... 90
Сверточная модель связывания TF ... 90
Доступность хроматина ... 93
РНК-интерференция ... 95
Заключение ... 98

Глава 7. Машинное обучение и микроскопия ... 99

Краткое введение в микроскопию ... 101
Современная оптическая микроскопия ... 102
Дифракционный предел ... 104
Электронная и атомно-силовая микроскопия ... 105
Микроскопия сверхвысокого разрешения ... 107
Глубокое обучение и дифракционный предел ... 109
Подготовка биологических препаратов для микроскопии ... 109
Окрашивание ... 109
Фиксация препаратов ... 110
Секционирование препаратов ... 111
Флуоресцентная микроскопия ... 111
Артефакты пробоподготовки ... 113
Применение глубокого обучения в микроскопии ... 114
Подсчет клеток ... 114
Клеточная сегментация ... 117
Вычислительные анализы ... 121
Заключение ... 121

Глава 8. Глубокое обучение в медицине ... 123

Компьютерная диагностика ... 123
Вероятностные диагнозы с байесовскими сетями ... 124

Данные электронных медицинских карт ... 126
В чем опасность больших баз данных ЭМК пациентов? ... 128
Глубокая радиология ... 129
Рентгенография и компьютерная томография ... 131
Гистология ... 133
Магниторезонансная томография ... 133
Модель глубокого обучения в качестве лечебного средства ... 134
Диабетическая ретинопатия ... 135
Перспективы глубокого обучения в медицине ... 139
Этические соображения ... 139
Потеря работы ... 140
Заключение ... 140

Глава 9. Генеративные модели ... 141

Вариационные автоэнкодеры ... 141
Генеративные состязательные сети ... 143
Применение генеративных моделей в науках о жизни ... 144
Генерация новых идей для соединений-прототипов ... 144
Разработка белков ... 145
Инструменты для научного поиска ... 145
Будущее генеративного моделирования ... 146
Работа с генеративными моделями ... 146
Анализ вывода генеративной модели ... 148
Заключение ... 151

Глава 10. Интерпретация глубоких моделей ... 154

Как объяснить предсказания? ... 154
Оптимизация входов ... 158
Прогнозирование неопределенности ... 161
Интерпретируемость, объяснимость и последствия для реального мира ... 165
Заключение ... 165

Глава 11. Практический пример виртуального скрининга ... 166

Подготовка набора данных для прогнозного моделирования ... 167
Обучение прогностической модели ... 172
Подготовка набора данных для прогнозирования ... 177
Применение прогностической модели ... 180
Заключение ... 186

Глава 12. Ожидания и перспективы ... 188

Медицинская диагностика ... 188
Персонализированная медицина ... 190
Фармацевтические исследования ... 191
Биологические исследования ... 193
Заключение ... 194

Колофон ... 195

Предметный указатель ... 196

Предисловие

Настало время, когда науки о жизни и данных соединились. Достижения в области робототехники и автоматики позволяют химикам и биологам получать огромное количество данных. Современный ученый за один день может сгенерировать больше данных, чем его предшественники два десятка лет назад могли бы собрать за всю карьеру. Эта способность быстро генерировать данные создала ряд новых научных проблем. Осталась позади эпоха, когда мы обрабатывали данные, загружая их в электронную таблицу и создавая пару графиков. Чтобы извлечь научные знания из новых огромных наборов данных, мы должны уметь выявлять и использовать неочевидные связи.

Одним из мощных инструментов для выявления закономерностей и взаимосвязей в данных является *глубокое обучение*, класс алгоритмов, которые произвели революцию в ряде областей, включая анализ изображений, языковой перевод и распознавание речи. Алгоритмы глубокого обучения превосходно зарекомендовали себя при выявлении и использовании шаблонов в больших наборах данных. По этим причинам глубокое обучение широко применяется во всех дисциплинах науки о жизни. В этой книге представлен обзор применения глубокого обучения в ряде областей, включая генетику, поиск лекарств и медицинскую диагностику. Обзор сопровождается примерами кода, которые помогают применить новые знания на практике и дают читателю отправную точку для будущих исследований и разработок.

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения.

Курсив – используется для смыслового выделения важных положений, новых терминов, имен команд и утилит, а также имен и расширений файлов и каталогов.

`Моноширинный шрифт` – используется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода.

`Моноширинный полужирный шрифт` – используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений.

`Моноширинный курсив` – используется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.

Такая пиктограмма обозначает совет или рекомендацию.

Такая пиктограмма обозначает указание или примечание общего характера.

Эта пиктограмма обозначает предупреждение или особое внимание к потенциально опасным объектам.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте **www.dmkpress.com** или **www.дмк.рф** на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Благодарности

Авторы хотели бы поблагодарить Николь Таш (Nicole Tache), нашего редактора в O'Reilly, а также технических рецензентов и бета-рецензентов за их ценный

вклад в книгу. Кроме того, мы хотели бы поблагодарить Карла Лесвинга (Karl Leswing) и Чжэньциня (Майкла) By (Zhenqin/Michael Wu) за их вклад в код, а также Джонни Израэли (Johnny Israeli) за ценные советы для главы по геномике.

Бхарат благодарит свою семью за поддержку и ободрение в течение многих долгих выходных и ночей, проведенных в работе над этой книгой.

Питер хотел бы поблагодарить свою жену за ее постоянную поддержку, а также многих коллег, от которых он так много узнал о машинном обучении.

Патрик благодарен своей жене Андреа и дочерям Эли и Мэдди за их любовь и поддержку. Он также благодарит прошлых и настоящих коллег из Vertex Pharmaceuticals и Relay Therapeutics, у которых он многому научился.

Наконец, мы хотим поблагодарить сообщество разработчиков программного обеспечения DeepChem за поддержку и консультации на протяжении работы над книгой.

Об авторах

Бхарат Рамсундар (Bharath Ramsundar) является соучредителем и техническим директором компании Datamined, занятой созданием больших наборов биологических данных. Эти наборы данных широко востребованы в связи с бумом ИИ в биотехнологиях. Бхарат также является ведущим разработчиком и создателем DeepChem.io, пакета с открытым исходным кодом, основанного на TensorFlow и нацеленного на повышение доступности глубокого обучения в области поиска лекарств, и соавтором пакета тестов MoleculeNet.

Бхарат получил степени бакалавра в Калифорнийском университете в Беркли по специальностям «Электроника и информатика» и «Математика» и был удостоен чести произнести прощальную речь перед выпускниками математического факультета. Недавно он защитил кандидатскую диссертацию по информатике в Стэнфордском университете (все, кроме формальной части) в группе Виджая Панде и при поддержке Hertz Fellowship, стипендиального фонда с самым строгим отбором аспирантов.

Питер Истман (Peter Eastman) разрабатывает программное обеспечение для биологов и химиков в отделе биотехнологий Стэнфордского университета. Он является ведущим автором OpenMM, инструментария для высокопроизводительного моделирования молекулярной динамики, и основным разработчиком DeepChem, пакета для глубокого машинного обучения в области химии, биологии и материаловедения. С 2000 года он является профессиональным инженером-программистом, в том числе вице-президентом по разработке программного обеспечения для Silicon Genetics – компании, занимающейся разработкой программного обеспечения для биоинформатики. Сейчас исследовательские интересы Питера сосредоточены на пересечении физики и глубокого обучения.

Пэт Уолтерс (Pat Walters) возглавляет группу по вычислительной информатике в Relay Therapeutics в Кембридже, штат Массачусетс. Его группа работает над новыми вычислительными методами, объединяющими компьютерное моделирование и экспериментальные данные в программах по разработке новых лекарственных препаратов. До прихода в Relay Therapeutics он более 20 лет проработал в Vertex Pharmaceuticals, где занимал должность генерального директора по моделированию и информатике.

Пэт является членом редакционно-консультативного совета «Журнала медицинской химии» и ранее занимал аналогичные должности в журналах «Молекулярная информатика» и «Бюллетень исследований и разработки лекарственных средств». Он продолжает играть активную роль в научном сообществе. Пэт был председателем конференции Гордона по компьютерной разработке лекарств 2017 года и сыграл важную роль в ряде профессиональных сообществ, включая базу данных разработчиков лекарственных средств (Drug Design Data Resource, D3R) и Ассоциацию американских химиков-разработчиков (American Chemical Society TDT initiative). Пэт получил докторскую степень по органической химии в университете Аризоны, где изучал применение искусственного интеллекта

в анализе молекулярных конформаций. До получения степени доктора он работал в Varian Instruments как химик и разработчик программного обеспечения. Пэт Уолтерс получил степень бакалавра по химии в Калифорнийском университете в Санта-Барбаре.

Виджай Панде (Vijay Pande), доктор философии, является генеральным партнером Andreessen Horowitz, где отвечает за инвестиции фирмы в области биологии и информатики, включая применение вычислений, машинного обучения и искусственного интеллекта в биологии и здравоохранении, а также инновационных научных технологий. Он работает адъюнкт-профессором факультета биотехнологий в Стэнфорде, где изучает применение компьютерных методов в медицине и биологии и консультирует студентов и аспирантов, что привело к появлению более двухсот публикаций, двух патентов и двух новых лекарственных препаратов.

Будучи предпринимателем, Виджай является основателем *Проекта распределенных вычислений Folding@Home по изучению заболеваний*, расширяющего границы применения компьютерных технологий, таких как распределенные системы, машинное обучение и экзотические компьютерные архитектуры, в биологии и медицине. Он занимается как фундаментальными исследованиями, так и разработкой новых методов лечения. Виджай также стал соучредителем Globavir Biosciences, где превратил свои научные достижения в Стэнфорде и Folding@Home в успешный стартап, открывая лекарства от лихорадки денге и лихорадки Эбола. В подростковом возрасте он был первым сотрудником стартапа видеоигр Naughty Dog Software, производителя популярной игровой франшизы Crash Bandicoot.

Глава 1

Глубокое обучение и науки о жизни

Существует много направлений, где могут проявить себя энтузиасты и эксперты по работе с данными, однако лишь немногие области могут сравниться с биомедицинскими исследованиями по фундаментальным последствиям применения больших данных. Появление современной медицины коренным образом изменило природу человеческого существования. За последние 20 лет мы увидели инновации, которые изменили жизнь множества людей. Впервые обнаруженный в 1981 году, ВИЧ/СПИД был смертельным заболеванием. Появление антиретровирусной терапии значительно увеличило продолжительность жизни больных ВИЧ в развитых странах. Другие болезни, такие как гепатит С, который десять лет назад считался в основном неизлечимым, теперь можно вылечить. Достижения в области генетики позволяют обнаруживать, и, надеемся, в скором времени, лечить, широкий спектр заболеваний. Инновации в диагностике и измерительной аппаратуре позволили врачам целенаправленно выявлять и контролировать заболевания в организме человека. Многие из этих прорывов случились и продолжают развиваться благодаря новым вычислительным методам.

Почему все говорят о глубоком обучении?

Алгоритмы машинного обучения теперь являются ключевым компонентом всех современных компьютерных технологий, от покупок в интернете до социальных сетей. Команды ученых-компьютерщиков разрабатывают алгоритмы, позволяющие цифровым помощникам, таким как Amazon Echo или Google Home, понимать речь. Достижения в области машинного обучения позволяют на лету выполнять перевод веб-страниц. Помимо влияния на повседневную жизнь, машинное обучение активно воздействует на многие области естественных наук и наук о жизни. Алгоритмы машинного обучения применяются ко всему, начиная с поиска новых галактик и заканчивая классификацией субатомных взаимодействий на Большом адронном коллайдере.

Одним из источников этих технологических достижений стало появление класса методов машинного обучения, известных как *глубокие нейронные сети*. Хотя технологические основы искусственных нейронных сетей были разработаны в 1950-х годах и усовершенствованы в 1980-х годах, истинная мощь этого метода не была полностью реализована до тех пор, пока за последнее десятилетие не

появились достаточно мощные компьютеры. Мы предоставим более полный обзор глубоких нейронных сетей в следующей главе, но хотим отдельно отметить некоторые из достижений, которые стали возможными благодаря применению глубокого обучения:

- *распознавание речи* в сотовых телефонах, компьютерах, телевизорах и других устройствах, подключенных к интернету, основано на глубоком обучении;
- *распознавание изображений* является ключевым компонентом беспилотных автомобилей, поиска в интернете и других приложений. Многие достижения в области глубокого обучения, которые привели к созданию потребительских приложений, в настоящее время используются в биомедицинских исследованиях. В одном из примеров исследователи используют глубокое обучение для классификации опухолевых клеток по типам и определения восприимчивости к лечению различными препаратами;
- *рекомендательные системы* стали ключевым компонентом сетевого взаимодействия. Такие компании, как Amazon, используют глубокое обучение, чтобы стимулировать дополнительные покупки при помощи подсказок «с этим товаром обычно покупают». Netflix использует аналогичный подход, чтобы рекомендовать фильмы, которые человек может захотеть посмотреть. Многие идеи, заложенные в основу этих рекомендательных систем, применяются для поиска молекул, которые могут стать отправной точкой в разработке новых лекарств;
- *языковой перевод* раньше выполняли очень сложные системы, основанные на правилах. За последние несколько лет системы перевода, основанные на глубоком обучении, далеко превзошли системы, основанные на ручном вводе правил. Аналогичные методы в настоящее время применяются для извлечения понятий из научной литературы и уведомления ученых о журнальных статьях, которые они могли пропустить.

Это лишь некоторые из инноваций, появившихся благодаря применению методов глубокого обучения. Мы живем в интересное время, когда происходит слияние массивов общедоступных научных данных и методов обработки этих данных. Те, кто способен комбинировать данные с новыми методами обучения по шаблонам в этих данных, могут добиться значительных научных достижений.

Современные науки о жизни – это науки о данных

Как упоминалось выше, фундаментальная природа наук о жизни изменилась. Доступность робототехники и микроэкспериментов привела к значительному увеличению объема извлекаемых экспериментальных данных. В 1980-х биолог проводил один эксперимент и получал один результат. Эти данные обычно можно обработать вручную с помощью карманного калькулятора. Если говорить о современной биологии, то у нас есть инструментарий, способный за день или два сгенерировать миллионы точек экспериментальных данных. Эксперименты, генерирующие огромные наборы данных, например секвенирование генов, стали недорогими и рутинными.

Успехи в секвенировании генов привели к созданию баз данных, которые связывают генетический код человека со множеством потенциальных заболеваний,

включая диабет, рак и генетические заболевания, такие как муковисцидоз. Используя вычислительные методы для извлечения и анализа данных, ученые совершенствуют понимание причин генетически обусловленных заболеваний и используют это понимание для разработки новых методов лечения.

Дисциплины, которые когда-то полагались главным образом на наблюдение за людьми, теперь используют наборы данных, просто не поддающиеся ручному анализу. Машинное обучение в настоящее время регулярно используется для классификации изображений клеток. Выходные данные этих моделей машинного обучения используются для обнаружения и классификации раковых опухолей и для оценки эффективности возможного лечения заболеваний.

Успехи в экспериментальных методах привели к созданию нескольких баз данных, которые каталогизируют структуры химических веществ и влияние этих веществ на широкий спектр биологических процессов или видов деятельности. Эти *структурно–функциональные взаимосвязи* (SAR, structure-activity relationship) составляют основу области, известной как *химическая информатика*, или *хемоинформатика*. В настоящее время ученые обрабатывают эти обширные наборы данных и используют их для построения *прогностических моделей*, на которых основаны новые методы разработки лекарств.

Новое поколение данных нуждается в новом поколении ученых, которые чувствуют себя комфортно как в лаборатории, так и за компьютером. Обладатели гибридных навыков способны увидеть закономерности и тенденции в больших наборах данных и сделать научные открытия завтрашнего дня.

О чем эта книга?

В первых нескольких главах мы даем обзор глубокого обучения и того, как его можно применять в науках о жизни. Мы начнем с *машинного обучения*, которое было определено как «наука и искусство программирования компьютеров, извлекающих знания из данных»[1].

Глава 2 дает краткое введение в разновидность машинного обучения, известную как *глубокое обучение*. Мы начнем с примера того, как глубокое обучение можно использовать для решения такой простой задачи, как линейная регрессия, и перейдем к более сложным моделям, которые обычно используются для решения реальных проблем в науках о жизни. Машинное обучение обычно сопровождается разделением начального набора данных на *обучающий набор* (training set), который используется для обучения модели, и *проверочный набор* (test set), который используется для проверки точности модели.

В главе 2 мы обсуждаем некоторые детали, связанные с обучением и проверкой прогностических моделей. После того как модель обучена, ее производительность обычно можно оптимизировать, изменяя ряд характеристик, известных как *гиперпараметры* (hyperparameters). В этой главе представлен обзор процесса оптимизации. Глубокое обучение – это не одиночный подход, а набор связанных методов. Глава 2 заканчивается знакомством с несколькими наиболее важными подходами к глубокому обучению.

---

[1] *Furbush J.* Machine Learning: A Quick and Simple Definition // **https://www.oreilly.com/ideas/machine-learning-a-quick-and-simple-definition**.

В главе 3 мы представляем *DeepChem*, библиотеку Python с открытым исходным кодом, которая была специально разработана для упрощения создания моделей глубокого обучения в области наук о жизни. Вслед за обзором DeepChem мы представляем наш первый пример программирования, демонстрирующий применение этой библиотеки. Мы создаем модель, предсказывающую токсичность молекул. Во втором примере программирования мы показываем, как DeepChem можно использовать для классификации изображений, что является обычной задачей в современной биологии. Как кратко упомянуто выше, глубокое обучение используется в различных приложениях *визуальной диагностики*, начиная от диагностики рака до выявления глаукомы. Из обсуждения конкретных приложений затем вытекает объяснение некоторых внутренних методов глубокого обучения.

В главе 4 представлен обзор того, как машинное обучение применяется к молекулам. Мы начинаем со знакомства с понятием молекул, составных частей окружающего нас мира. Хотя молекулы в какой-то мере можно считать аналогом строительных кирпичей, они являются гибкими и демонстрируют динамическое поведение. Чтобы охарактеризовать молекулы с помощью вычислительных методов, таких как глубокое обучение, нам нужно найти способ представления молекул в компьютере. Компьютерная кодировка молекул похожа на представление изображения в виде набора пикселей. Во второй половине главы 4 мы описываем ряд способов представления молекул и их применение для построения моделей глубокого обучения.

Глава 5 содержит введение в *биофизику* – науку, применяющую законы физики к биологическим явлениям. Мы начнем с обсуждения белков, своеобразных молекулярных машин, благодаря которым существует белковая форма жизни. Прогнозирование воздействия лекарств на организм невозможно без понимания эффектов их взаимодействия с белками. Чтобы понять эти эффекты, мы начнем с обзора того, как устроены белки и как различаются их структуры. Белки – это объекты, чья трехмерная структура определяет их биологическую функцию. Для того чтобы модель машинного обучения могла предсказать влияние молекулы лекарства на функцию белка, мы должны представить трехмерную структуру белка в форме, понятной для программы машинного обучения. Во второй половине главы 5 мы рассматриваем несколько способов представления структур белка. Опираясь на эти знания, мы представим еще один пример кода, где глубокое обучение применяется для предсказания эффекта взаимодействия молекулы лекарства с белком.

Генетика стала ключевым компонентом современной медицины. Генетическое секвенирование опухолей позволило персонализировать лечение рака и может произвести революцию в медицине. Секвенирование генов, которое раньше было сложным процессом, требующим огромных инвестиций, теперь стало обычным делом и может проводиться где угодно. Мы уже достигли этапа, когда владельцы собак могут провести недорогие генетические тесты для определения происхождения своего питомца. В главе 6 мы даем обзор генетики и геномики, начиная со знакомства с молекулами ДНК и РНК, важнейшими шаблонами, для производства белков. Недавние открытия показали, что взаимодействия ДНК и РНК протекают намного сложнее, чем считалось изначально. Во второй половине главы 6 мы представляем несколько примеров кода, предсказывающего влияние различных факторов на взаимодействие ДНК и РНК.

Ранее в этой главе мы упоминали о выдающихся успехах, которые были достигнуты благодаря применению глубокого обучения для анализа биологических и медицинских изображений. Многие из изучаемых в этих экспериментах явлений слишком малы, чтобы их можно было наблюдать человеческим глазом. Изображения для последующего анализа методами глубокого обучения получают при помощи микроскопа. В главе 7 представлен обзор микроскопии в ее многочисленных формах, от простого светового микроскопа, который мы все использовали в школе, до сложных приборов, способных получать изображения одиночных атомов. Эта глава также охватывает некоторые ограничения современных подходов и знакомит с экспериментальными источниками изображений, на которых строятся модели глубокого обучения.

Одной из областей, в которой раскрываются огромные перспективы, является применение глубокого обучения для медицинской диагностики. Медицина невероятно сложна, и ни один врач не способен вобрать в себя все имеющиеся медицинские знания. В идеале система машинного обучения могла бы изучить всю медицинскую литературу в мире и помогать врачам ставить диагнозы. Хотя мы еще далеки от этого уровня, но уже есть успехи. Глава 8 начинается с истории использования методов машинного обучения в медицинской диагностике и показывает переход от ручного кодирования правил к статистическому анализу медицинских данных. Как и во многих обсуждаемых нами темах, ключевой проблемой является представление медицинской информации в компьютерном формате, понятном программам машинного обучения. В этой главе мы представляем введение в электронные медицинские карты и некоторые проблемы, связанные с этими записями. Во многих случаях медицинские изображения бывают очень сложными, и анализ и интерпретация этих изображений вызывают затруднения даже у квалифицированных специалистов. В этих случаях глубокое обучение может улучшить навыки эксперта, классифицируя изображения и выявляя ключевые особенности. Глава 8 завершается рядом примеров того, как глубокое обучение используется для анализа изображений в различных областях медицины.

Как мы упоминали ранее, машинное обучение становится ключевым инструментом для поиска новых лекарств. Ученые используют модели глубокого обучения, чтобы оценить взаимодействия между молекулами лекарств и белками. Эти взаимодействия могут вызывать биологический отклик, оказывающий терапевтическое воздействие на пациента. Модели, которые мы упоминали до сих пор, являются *дискриминантными моделями*.

Исходя из характеристик молекулы, модель генерирует прогноз некоторого свойства вещества. Описание молекулы может быть получено из обширной базы данных существующих молекул или родиться в воображении ученого. Но что, если мы не будем полагаться на существующие описания или собственную фантазию и разработаем компьютерную программу, способную «изобретать» новые молекулы? Глава 9 представляет *генеративную модель*, которая сначала обучается на наборе существующих молекул, а затем применяется для генерации новых молекул. Модель глубокого обучения, которая генерирует эти молекулы, также может зависеть от других моделей, предсказывающих активность новых молекул.

До сих пор мы обсуждали модели глубокого обучения как «черные ящики». Мы запускаем модель с набором входных данных, и она генерирует прогноз без объяснения того, как или почему прогноз был сгенерирован. Во многих ситуациях это

неприемлемо. Если у нас есть модель глубокого обучения для медицинской диагностики, нам нужно понимать причины диагноза. Объяснение причин диагноза даст врачу больше уверенности в прогнозе, а также может повлиять на решение о лечении. У моделей глубокого обучения есть исторический недостаток – прогнозы даже самых надежных моделей трудно поддаются логическому обоснованию. В настоящее время разрабатывается ряд методов, позволяющих пользователям лучше понять факторы, влияющие на прогноз. В главе 10 представлен обзор некоторых методов, способствующих пониманию причин прогноза. Другим важным аспектом *предсказательного моделирования* является *точность предсказаний* модели. Знание точности модели помогает решить, насколько можно полагаться на эту модель. Учитывая, что машинное обучение может использоваться для постановки жизненно важных диагнозов, оценка точности модели имеет решающее значение. В последнем разделе главы 10 представлен обзор некоторых методов оценки точности предсказания модели.

В главе 11 мы представляем практический пример использования DeepChem. В этом примере мы используем метод, известный как *виртуальный скрининг*, чтобы определить потенциальные отправные точки для поиска новых лекарств. Создание новых лекарств – это сложный процесс, который часто начинается с техники, известной как скрининг. Он применяется для поиска молекул, которые могут стать основой для новых лекарств. Скрининг может проводиться экспериментально, когда миллионы молекул тестируются в миниатюрных биологических тестах (так называемых *пробах*), или на компьютере с использованием виртуального скрининга. В методе виртуального скрининга модель обучается на наборе известных лекарств или других биологически активных молекул. Затем эта модель используется для прогнозирования активности большого количества новых молекул. Благодаря высокому быстродействию машинного обучения за несколько дней вычислений удается обработать сотни миллионов молекул.

Книга заканчивается главой 12 «Ожидания и перспективы». В этой главе рассматривается сегодняшнее положение дел и перспективы применения глубокого обучения в науках о жизни. Обсуждается ряд текущих проблем, включая доступность и качество наборов данных. Мы также выделяем перспективы и подводные камни в ряде других областей, включая диагностику, персонализированную медицину, фармацевтику и исследования в области биологии.

Глава 2

Введение в глубокое обучение

В этой главе представлены основные принципы глубокого обучения. Если вы хорошо знакомы с глубоким обучением, то можете пропустить эту главу и перейти к следующей. Новичкам в области глубокого обучения следует внимательно изучить главу, поскольку материал, который она охватывает, будет важен для понимания остальной части книги.

Решение большинства проблем, которые мы обсудим, сводится к созданию математической функции общего вида

$$\mathbf{y} = f(\mathbf{x}).$$

Обратите внимание, что **x** и **y** выделены жирным шрифтом. Это означает, что они являются *векторами*. Функция может принимать тысячи или даже миллионы чисел в качестве входных данных, и она может выдавать столь же много чисел в качестве выходных данных. Вот несколько примеров функций, которые вы могли бы создать:

- **x** содержит цвета всех пикселей изображения. $f(\mathbf{x})$ равно 1, если изображение содержит кошку, и 0, если это не так;
- то же, что и выше, однако $f(\mathbf{x})$ должно быть *числовым вектором*. Первый элемент указывает, содержит ли изображение кошку, второй – содержит ли оно собаку, третий – содержит ли оно самолет, и так для тысяч разновидностей объектов;
- **x** содержит последовательность ДНК хромосомы, а **y** должен быть вектором, длина которого равна числу оснований в хромосоме. Каждый элемент должен равняться 1, если это основание является частью области, кодирующей белок, и 0, если это не так;
- **x** описывает структуру молекулы. Мы обсудим различные способы компьютерного представления молекул в последующих главах. **y** должен быть вектором, где каждый элемент описывает некоторое физическое свойство молекулы, например растворимость в воде, силу связи с другой молекулой, и т. д.

Как видите, функция $f(\mathbf{x})$ может быть очень и очень сложной! Обычно она принимает длинный вектор в качестве входных данных и пытается извлечь из него информацию, которая совсем не очевидна, если взглянуть на входные числа.

Традиционный подход к решению этой проблемы заключается в разработке функции вручную. При таком подходе вы бы начали с анализа проблемы. Какие сочетания пикселей с наибольшей вероятностью указывают на присутствие кошки? Какие паттерны ДНК указывают на отличие кодирующих участков от некодирующих? Вы должны написать компьютерный код для выделения определенных признаков, а затем попытаться определить комбинации признаков, которые надежно обеспечивают желаемый результат. Это медленный и трудоемкий процесс, который сильно зависит от опыта человека, выполняющего анализ данных.

Машинное обучение использует совершенно другой подход. Вместо того чтобы разрабатывать функцию вручную, вы позволяете компьютеру *обучить* свою собственную функцию на основе данных. Вы берете тысячи или миллионы изображений и помечаете те из них, на которых есть кошка. Затем вы предоставляете эти *обучающие данные* компьютеру и позволяете ему искать функцию, которая стремится к 1 для изображений с кошками и к 0 для изображений, на которых кошки нет.

Что значит «позволить компьютеру искать функцию»? В общем вы создаете *модель*, которая определяет некоторый большой класс функций. Модель включает в себя *параметры* – переменные, которые могут принимать любое значение. Выбирая значения параметров, вы тем самым выбираете конкретную функцию из всего множества функций в классе. Задача компьютера заключается в подборе значений параметров. Компьютер пытается найти такие значения, чтобы при подаче на вход модели обучающих данных ее выходные данные были как можно ближе к соответствующим целевым значениям.

Линейные модели

Одна из самых простых моделей, которые вы можете рассмотреть, – это *линейная модель*:

$$\mathbf{y} = M\mathbf{x} + \mathbf{b}.$$

В этом уравнении $\mathbf{M}$ представляет собой матрицу (иногда называемую «весами»), а $\mathbf{b}$ – это вектор (называемый «отклонениями»). Их размеры определяются количеством входных и выходных значений. Если $\mathbf{x}$ имеет длину T и вы хотите, чтобы $\mathbf{y}$ имел длину S, то $\mathbf{M}$ будет матрицей $S \times T$, а $\mathbf{b}$ будет вектором длины S. Вместе они составляют параметры модели. Это уравнение просто говорит о том, что каждый выходной компонент является линейной комбинацией входных компонентов. Задавая параметры $\mathbf{M}$ и $\mathbf{b}$, вы можете выбрать для каждого компонента любую линейную комбинацию, которую пожелаете.

Это была одна из самых ранних моделей машинного обучения. Она была предложена еще в 1957 году под названием «*персептрон*»[1] (perceptron). Название представляет собой яркий пример научного маркетинга. В нем звучит научная фантастика, и оно обещает удивительные вещи, хотя на самом деле это не что иное, как линейное преобразование. Во всяком случае, названию удалось продержаться более полувека.

[1] На русском языке иногда пишут «перцептрон». – *Прим. перев.*

Линейная модель очень легко записывается в общем виде. Она имеет одну и ту же форму независимо от того, к какой проблеме она относится. Единственные различия между линейными моделями – это длина входного и выходного векторов. В данном случае обучение – это лишь вопрос выбора значений параметров, что можно сделать простым способом с помощью общих алгоритмов. С точки зрения машинного обучения это идеальный подход, при котором модель и алгоритмы не зависят от решаемой проблемы. Для автоматического определения параметров достаточно лишь предоставить обучающие данные, и модель будет преобразована в функцию, которая решает вашу проблему.

К сожалению, линейные модели также очень ограничены. Как показано на рис. 2.1, линейная модель (уравнение первой степени, описывающее прямую линию) просто не может соответствовать большинству реальных наборов данных. Ситуация становится еще хуже, когда вы переходите к очень многомерным данным. Никакая линейная комбинация пикселей в изображении не будет надежно определять, содержит ли изображение кошку. Задача распознавания изображений требует гораздо более сложной нелинейной модели. Фактически любая модель, которая решает проблему поиска объектов, обязательно будет очень сложной и очень нелинейной. Но как мы можем определить нелинейную модель в общем виде? Пространство всех возможных нелинейных функций бесконечно велико. Каким образом определить модель так, чтобы, просто выбирая значения параметров, мы могли создать практически любую нелинейную функцию, которая нам когда-либо понадобится?

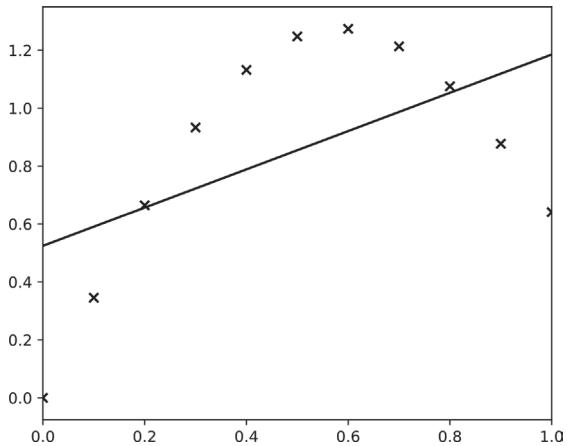


Рис. 2.1 ❖ Линейная модель не может представлять точки данных, лежащие на кривой. Для этого нужна нелинейная модель

Многослойные персептроны

Простой подход заключается в последовательном объединении нескольких линейных преобразований. Например, мы можем записать двойное преобразование в следующем виде:

$$\mathbf{y} = \mathbf{M}_2\varphi(\mathbf{M}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2.$$

Постарайтесь вникнуть в последовательность действий. Мы начинаем с обычного линейного преобразования $\mathbf{M}_1\mathbf{x} + \mathbf{b}_1$. Затем пропускаем результат преобразования через нелинейную функцию $\varphi(x)$ и к полученному выходу применяем второе линейное преобразование. Функция $\varphi(x)$, известная как *функция активации*, является важной частью модели. Без нее модель все равно осталась бы линейной и не более мощной, чем предыдущая. Ведь линейная комбинация линейных комбинаций все равно остается не более чем линейной комбинацией исходных входов! Внося нелинейность, мы даем возможность модели исследовать гораздо более широкий диапазон функций.

Мы можем не останавливаться на двух линейных преобразованиях и выстроить их в последовательность произвольной длины:

$$\mathbf{h}_1 = \varphi_1(\mathbf{M}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \varphi_2(\mathbf{M}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\ldots$$
$$\mathbf{h}_{n-1} = \varphi_{n-1}(\mathbf{M}_{n-1}\mathbf{h}_{n-2} + \mathbf{b}_{n-1})$$
$$\mathbf{y} = \varphi_n(\mathbf{M}_n\mathbf{h}_{n-1} + \mathbf{b}_n).$$

Такая модель называется *многослойным персептроном* (multilayer perceptron, MLP). Промежуточные шаги h_i называются *скрытыми слоями*. Их название отражает тот факт, что они не являются ни входными данными, ни выходными данными, а являются лишь промежуточными значениями, используемыми в процессе вычисления результата. Также обратите внимание, что мы добавили индекс к каждой функции $\varphi(x)$, указывая на то, что разные слои могут использовать разные нелинейности.

Эту последовательность вычислений можно визуально представить в виде *стека слоев*, как показано на рис. 2.2. Каждый слой соответствует линейному преобразованию с последующей нелинейностью. Информация перемещается с одного уровня на другой, а выход одного уровня становится входом для следующего. Каждый слой имеет свой собственный набор параметров, которые определяют, как его выход рассчитывается на основе его входных данных.

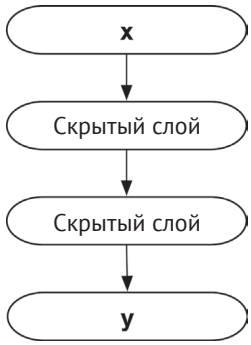


Рис. 2.2 ❖ Многослойный персептрон представлен как стек слоев с информацией, перетекающей из одного слоя в другой

Многослойные персептроны и их вариации также называют *нейронными сетями*. Название отражает сходство между машинным обучением и нейробиологией. Биологический нейрон соединяется со многими другими нейронами. Он получает от них сигналы, складывает их вместе, а затем отправляет свои собственные сигналы другим нейронам в зависимости от результата. В очень грубом приближении можно считать, что MLP работают так же, как нейроны в вашем мозгу.

Какой должна быть функция активации $\varphi(x)$? Неожиданный ответ заключается в том, что это в целом не имеет значения! Конечно, это не совсем так, но значение не столь велико, как вы могли ожидать. Подойдет почти любая монотонная и достаточно плавная непрерывная функция. За долгие годы было опробовано множество различных функций, и хотя некоторые из них работают лучше, чем другие, почти все они дают приемлемые результаты.

На сегодняшний день наиболее популярной функцией активации считается *выпрямленная линейная функция* (Rectified Linear Unit, ReLU) $\varphi(x) = \max(0, x)$. Если вы не знаете, какую функцию выбрать, она будет хорошим вариантом по умолчанию. К другим распространенным вариантам относятся *гиперболический тангенс* $\tanh(x)$ и *логистическая сигмоида* $\varphi(x) = 1/(1 + e^{-x})$. Все эти функции показаны на рис. 2.3.

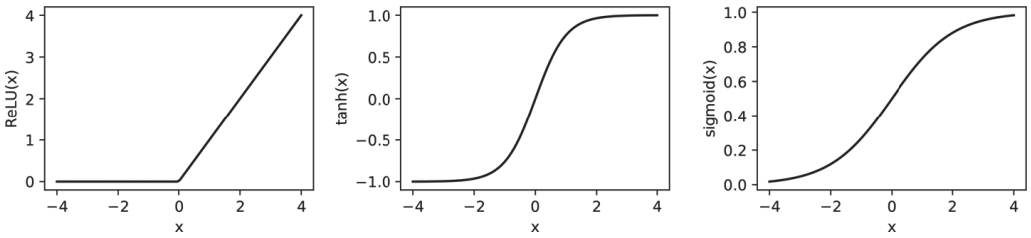


Рис. 2.3 ❖ Три обобщенные функции активации: выпрямленная линейная (ReLU), гиперболический тангенс и логистическая сигмоида

Мы также должны выбрать два других свойства MLP: его *ширину* и *глубину*. Ширина означает размер скрытых слоев. С простой линейной моделью у нас не было выбора. Размеры **M** и **b** были полностью определены исходя из длины **x** и **y**. Другое дело – скрытые слои. Для каждого $\mathbf{h}_i$ мы можем выбрать любую длину, какую пожелаем. В зависимости от задачи $\mathbf{h}_i$ могут быть намного больше или намного меньше, чем входные и выходные векторы.

Глубина относится к числу слоев в модели. Модель только с одним скрытым слоем называется *неглубокой* (shallow model). Модель со многими скрытыми слоями описывается как *глубокая* (deep model). Отсюда, по сути, происходит термин «глубокое обучение», означающий машинное обучение с использованием многослойных моделей.

Выбор количества и ширины слоев в модели – это в равной мере искусство и наука. Или, говоря более формально: «Это все еще активная область исследований». Часто все сводится к тому, чтобы попробовать множество комбинаций и посмотреть, какая из них лучше сработает. Однако есть несколько общих прин-

ципов, которые послужат руководством или, по крайней мере, помогут вам понять результаты задним числом.

1. MLP с одним скрытым слоем является *универсальным аппроксиматором*. Это означает, что он может аппроксимировать вообще любую функцию (в определенных достаточно разумных пределах). В каком-то смысле вам никогда не понадобится более одного скрытого слоя. Теоретически этого уже достаточно, чтобы воспроизвести любую функцию. К сожалению, этот результат сопровождается серьезным побочным эффектом: точность аппроксимации зависит от *ширины* скрытого слоя, и вам может потребоваться очень широкий слой, чтобы получить достаточную точность для текущей задачи. Это рассуждение подводит нас ко второму принципу.
2. Глубокие модели, как правило, требуют меньше параметров, чем неглубокие. Это утверждение намеренно сформулировано несколько расплывчато. Для конкретных случаев можно доказать и более строгие утверждения, но в общем можно сказать так: каждая проблема требует модели с определенной глубиной для эффективного достижения приемлемой точности. И как только глубина уменьшается ниже этого предела, необходимая ширина слоев (и следовательно, общее количество параметров) резко возрастает. Можно предположить, что глубокие модели всегда лучше. К сожалению, это частично противоречит третьему принципу.
3. Глубокие модели, как правило, труднее обучать, чем неглубокие. Примерно до 2007 года большинство моделей машинного обучения оставались неглубокими. Теоретические преимущества глубоких моделей были известны, но исследователям обычно не удавалось их обучать. С тех пор появились улучшенные алгоритмы обучения, новые типы моделей, которые легче обучать, и, конечно, более быстрые компьютеры в сочетании с большими наборами обучающих данных. Эти достижения повысили полезность глубоких моделей и дали начало «глубокому обучению» как самостоятельной отрасли знаний. Тем не менее, несмотря на эти достижения, общий принцип остается прежним: чем глубже модель, тем труднее ее обучать.

Обучение модели

Наверное, у вас возник закономерный вопрос: как же нам обучать модель? MLP предоставляют нам общую модель, которая может использоваться для любой проблемы. Мы обсудим другие, более специализированные типы моделей чуть позже. Теперь мы хотим, чтобы аналогичный общий алгоритм нашел оптимальные значения параметров модели для данной задачи. Как мы это сделаем?

Конечно, первое, что вам нужно, – это *обучающий набор*. Он должен состоять из большого числа пар (**x**, **y**), также известных как *выборки*[1] (samples). Каждый набор определяет входные данные для модели и указывает, что ожидается на выходе модели при заданных входных данных. Например, обучающая выборка может

[1] В публикациях на русском языке иногда встречается взаимная перестановка терминов: обучающий набор называют обучающей выборкой, а пару (**x**, **y**) называют выборкой или прецедентом. Нужно просто уяснить, что независимо от терминов модель учится на большом количестве соответствий «вход-выход». – *Прим. перев.*

представлять собой набор изображений вместе с метками, указывающими, содержит ли данное изображение кошку или нет.

Далее необходимо определить *функцию потерь* $L(\mathbf{y}, \hat{\mathbf{y}})$, где $\mathbf{y}$ – это фактический результат модели, а $\hat{\mathbf{y}}$ – целевое значение, указанное в обучающем наборе. Таким образом вы измеряете, насколько хорошо модель воспроизводит данные обучения. Затем производится усреднение по всем выборкам в обучающем наборе:

$$\text{Средние потери} = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{y}_i, \hat{\mathbf{y}}_i).$$

Значение $L(\mathbf{y}, \hat{\mathbf{y}})$ должно быть маленьким, когда аргументы функции близки друг к другу, и большим, когда они далеко друг от друга. Другими словами, мы берем каждую выборку в обучающем наборе, используем ее в качестве входных данных для модели и смотрим, насколько выходной сигнал отличается от целевого значения. Затем усредняем разницу по всему обучающему набору.

Для каждой конкретной проблемы должна быть выбрана соответствующая функция потерь. Распространенным вариантом является *евклидово расстояние* (также известное как расстояние L_2):

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \sqrt{\sum_i (y_i + \hat{y}_i)^2},$$

где y_i означает i-й компонент вектора $\mathbf{y}$.

Когда $\mathbf{y}$ представляет распределение вероятности, обычно применяется *перекрестная энтропия* (cross entropy) $L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i$. Возможны и другие варианты, поскольку универсального «лучшего» выбора не существует. Это зависит от особенностей вашей проблемы.

Теперь, когда у нас есть способ измерить, насколько хорошо работает модель, нам нужен способ ее улучшить. Мы хотим найти значения параметров, которые минимизируют среднюю потерю по обучающему набору. Есть много способов минимизации средней потери, но в большинстве случаев используют один из вариантов алгоритма *градиентного спуска*. Пусть θ представляет собой набор всех параметров в модели. Градиентный спуск представляет собой ряд небольших шагов

$$\theta \leftarrow \theta - \epsilon \frac{\partial}{\partial \theta}(L),$$

где L – средняя потеря за тренировочный набор. Каждый шаг перемещает модель на небольшое расстояние в направлении «вниз» и немного меняет каждый из параметров модели для уменьшения средней потери. Если звезды вам благоволят, а Луна вошла в правильную фазу, то в конечном итоге вы получите параметры, которые помогут решить вашу проблему. Коэффициент ϵ называется *скоростью обучения* (learning rate) и определяет степень изменения параметров на каждом шаге. Его нужно выбирать очень осторожно: слишком маленькое значение приведет к тому, что обучение будет очень медленным, а слишком большое значение вообще не позволит модели обучиться.

Этот алгоритм действительно работает, но у него есть серьезная проблема. Для каждого шага градиентного спуска вам нужно пройтись по каждой выборке в обучающем наборе. Это означает, что время, необходимое для обучения модели, пропорционально размеру обучающего набора! Предположим следующее:

- у вас есть один миллион выборок в обучающем наборе;
- для вычисления градиента потерь для одной выборки требуется один миллион операций;
- для поиска хорошей модели требуется один миллион шагов.

Все эти числа довольно типичны для реальных приложений глубокого обучения. В таком случае для обучения оптимальной модели потребуется один *квинтиллион* (10^{18}) операций. Это занимает довольно много времени даже на быстром компьютере.

К счастью, есть лучшее решение: оценить L путем усреднения по гораздо меньшему количеству выборок. На этом подходе основан алгоритм *стохастического градиентного спуска* (stochastic gradient descent, SGD). Для каждого шага из учебного набора берется небольшой набор образцов, так называемый *пакет* (batch). Затем вычисляется градиент функции потерь, усредненный только по выборкам в пакете. Мы можем рассматривать усреднение по пакету как оценку того, что могли бы получить усреднением по всему тренировочному набору, хотя такая оценка бывает очень зашумленной. Мы выполняем один шаг градиентного спуска, затем выбираем новую партию образцов для следующего шага.

Этот алгоритм обычно работает намного быстрее. Время, необходимое для каждого шага, не зависит от размера обучающего набора и определяется только размером пакета, который может быть довольно небольшим, зачастую не более 100 выборок. Недостаток метода заключается в том, что каждый шаг делает менее эффективную работу по сокращению потерь, потому что он основан на зашумленной оценке градиента, а не на истинном градиенте. Тем не менее применение SGD значительно сокращает время обучения в целом.

Большинство алгоритмов оптимизации, используемых в глубоком обучении, основано на улучшенных модификациях SGD. К счастью, вы можете рассматривать эти алгоритмы как «черные ящики» и доверять им делать свою работу, не вникая в детали. Два самых популярных алгоритма, используемых сегодня, называются Adam и RMSProp. Если вы сомневаетесь в том, какой алгоритм использовать, выбирайте для начала любой из этих двух.

Проверка модели

Предположим, вы сделали все, что описано выше. Вы собрали большой набор обучающих данных. Вы выбрали модель, затем запустили алгоритм обучения, пока значение функции потерь L не стало очень маленьким. Поздравляем, теперь у вас есть функция, которая решает вашу проблему!

Правильно?

Простите, но это не так просто! Вы можете быть точно уверены лишь в том, что модель хорошо работает с *обучающими данными*. Вы надеетесь, что она будет столь же хорошо работать с другими данными, но, конечно, не можете рассчитывать на это. Теперь вам нужно проверить модель, чтобы увидеть, работает ли она с данными, на которых специально не обучалась.

Для этого вам понадобится второй набор данных, который называется *оценочным набором* (evaluation set). Он, как и обучающий набор, состоит из множества пар ($\mathbf{x}$, $\mathbf{y}$), но у них не должно быть общих выборок. Вы обучаете модель на обучаю-

щем наборе, а затем оцениваете ее качество на оценочном наборе. Это подводит нас к одному из самых важных принципов машинного обучения.

Ни в коем случае не используйте оценочный набор при разработке или обучении модели.

На самом деле лучше всего, если вы даже краем глаза не посмотрите на данные в оценочном наборе. Единственное, для чего нужен оценочный набор, – это протестировать полностью обученную модель, чтобы выяснить, насколько хорошо она работает. Если вы позволите оценочному набору каким-либо образом влиять на модель, то рискуете получить модель, которая работает лучше на оценочном наборе, чем на других данных, которые не были задействованы при создании модели. Он перестает быть настоящим оценочным набором и превращается в еще один обучающий набор.

Это ограничение связано с математической концепцией *переобучения* (overfitting). В идеале обучающие данные должны охватывать набор всех входных данных, с которыми когда-либо столкнется модель, то есть быть практически бесконечными. Но в реальной жизни это невозможно. Вы можете сформировать конечный набор обучающих выборок, обучить на них модель и надеяться, что она найдет общую стратегию, которая одинаково хорошо работает на других выборках.

Переобучение означает, что модель слишком хорошо усвоила мелкие специфические особенности обучающих выборок и работает на них лучше, чем на незнакомых выборках.

Регуляризация

Переобучение является серьезной проблемой для тех, кто применяет машинное обучение. Поэтому не следует удивляться множеству методов, позволяющих избежать переобучения. В целом эти методы известны как *регуляризация* (regularization). Цель любого метода регуляризации – избежать переобучения и создать обученную модель, которая хорошо работает на любых входных данных, а не только на конкретном обучающем наборе.

Прежде чем мы обсудим конкретные методы регуляризации, необходимо понять два важных момента.

Во-первых, лучшим способом избежать переобучения *почти всегда* является расширение обучающего набора. Чем больше ваш обучающий набор, тем лучше он представляет «истинное» распределение данных и тем меньше вероятность того, что алгоритм обучения чрезмерно сосредоточится на небольшом обучающем наборе. Конечно, это иногда невозможно. Может быть, у вас просто нет возможности получить больше данных, или сбор данных обходится слишком дорого. В этом случае вам нужно приложить максимум усилий к имеющимся данным, а если возникнет переобучение, вам придется использовать регуляризацию. Но увеличение набора данных приводит к лучшему результату, чем регуляризация.

Во-вторых, не существует универсального «лучшего» способа регуляризации. Все зависит от проблемы. В конце концов, алгоритм обучения не знает, что он переобучается. Все, что он знает, – это обучающие данные. Он не знает, чем истинные данные отличаются от обучающих данных, поэтому лучшее, что он может

сделать, – создать модель, которая хорошо работает на обучающем наборе. Если это не то, чего вы хотели, это ваша проблема.

Суть любого метода регуляризации состоит в смещении процесса обучения с целью обеспечить преобладание одних типов моделей над другими. Вы делаете предположения о том, какими свойствами должна обладать «хорошая» модель и чем она отличается от переобученной модели, а затем указываете алгоритму обучения отдавать предпочтение моделям с нужными свойствами. Конечно, эти предположения часто являются интуитивными. Не всегда очевидно, из каких соображений вы исходите, выбирая конкретный метод регуляризации. Но, так или иначе, это приходится делать.

Один из самых простых методов регуляризации сводится к обучению модели на меньшем количестве шагов. На ранних этапах обучения модель склонна обнаруживать обобщенные свойства обучающих данных, которые, скорее, относятся к истинному распределению. Чем дольше модель работает, тем больше вероятность, что она начнет собирать мелкие незначительные детали конкретных обучающих выборок. Ограничивая количество обучающих шагов, вы даете модели меньше возможностей для обучения. Но при этом вы предполагаете, что «хорошие» значения параметров не слишком сильно отличаются от тех значений, которые найдены на раннем этапе обучения.

Другой метод заключается в ограничении размаха значений параметров в модели. Например, вы можете добавить к функции потерь член, пропорциональный $|\theta|^2$, где θ – вектор, содержащий все параметры модели. Мы делаем это из соображения, что значения «хороших» параметров не должны быть больше, чем необходимо. Переобучение часто (хотя и не всегда) приводит к тому, что некоторые параметры становятся очень большими.

Очень популярный метод регуляризации называется *исключением*, или *выпадением* (dropout). На первый взгляд этот метод может показаться нелепым, но на самом деле он работает на удивление хорошо. Для каждого скрытого слоя в модели случайным образом выбирают подмножество элементов выходного вектора h_i и обнуляют их. На каждом шаге градиентного спуска выбирают разные случайные подмножества элементов. Может показаться, что это просто сломает модель. Как можно ожидать, что случайное обнуление внутренних вычислений пойдет модели на пользу? Мы не будем углубляться в достаточно сложное математическое обоснование работоспособности исключения. Грубо говоря, мы исходим из того, что никакие индивидуальные вычисления внутри модели не должны быть чересчур важными. Эффективная модель должна продолжать работать, если из нее удалить несколько произвольных индивидуальных вычислений. Это заставляет модель изучать избыточные и хорошо распределенные представления данных, которые делают переобучение маловероятным. Если вы не уверены, какой метод регуляризации использовать, первым делом стоит попробовать исключение.

Оптимизация гиперпараметров

Наверное, вы уже заметили, что даже при использовании предположительно «универсальной» модели с «универсальным» алгоритмом обучения остается много параметров для выбора и настройки. Например, мы можем выбирать:

- количество слоев в модели;
- ширину каждого слоя;

- количество шагов обучения;
- скорость обучения;
- долю исключаемых элементов.

Подобные параметры называются *гиперпараметрами*. Гиперпараметр – это любой аспект модели или алгоритма обучения, который должен быть определен заранее, а не изучен алгоритмом обучения. Но как вы должны их определить? И разве смысл машинного обучения не состоит в том, чтобы автоматически выбирать настройки на основе данных?

Эти вопросы подводят нас к теме *оптимизации гиперпараметров*. Самый простой способ оптимизации заключается в переборе множества значений для каждого гиперпараметра и поиске наилучшего сочетания. При большом количестве гиперпараметров полный перебор обходится слишком дорого, поэтому существуют более рациональные методы оптимизации. Впрочем, основная идея остается прежней: пробовать разные комбинации и смотреть, какая из них работает лучше всего.

Но как можно сказать, что работает лучше всего? Проще всего посмотреть, какое сочетание параметров дает наименьшее значение функции потерь (или какой-либо другой меры точности) на обучающем наборе. Но это не то, что нас волнует. Мы хотим минимизировать ошибку на оценочном, а не на обучающем наборе. Это особенно важно для гиперпараметров, влияющих на регуляризацию, таких как частота исключений. Низкая ошибка обучающего набора может означать, что модель переобучена и учитывает мелкие детали обучающих данных. Поэтому мы хотим перебрать множество значений гиперпараметров, а затем использовать те сочетания, которые сводят к минимуму потери в оценочном наборе.

Но мы не должны так поступать! Запомните: нельзя использовать оценочный набор при разработке или обучении модели. Его задача – показать вам, насколько хорошо модель будет работать с новыми данными, которые она никогда не видела раньше. Тот факт, что определенный набор гиперпараметров работает лучше всего на оценочном наборе, отнюдь не гарантирует, что эти значения *всегда* будут работать лучше всего. Мы не должны допускать, чтобы оценочный набор влиял на модель, иначе он перестанет быть объективным оценочным набором.

Решение состоит в том, чтобы создать еще один набор данных, который называется *проверочным набором* (validation set). Он не должен пересекаться по выборкам ни с обучающим набором, ни с оценочным набором. Полная процедура теперь работает следующим образом:

- для каждого набора значений гиперпараметров обучите модель на обучающем наборе, а затем вычислите потери на проверочном наборе;
- найдя сочетание гиперпараметров, при котором получается минимум потерь на проверочном наборе, примите его в качестве параметров окончательной модели;
- опробуйте эту окончательную модель на оценочном наборе, чтобы получить объективную оценку того, насколько хорошо она работает.

Другие типы моделей

Вам осталось принять еще одно решение, и это само по себе огромный вопрос: какую модель использовать. Ранее в этой главе мы представили многослойные персептроны. Их преимущество заключается в том, что они представляют собой общий класс моделей, которые можно применять для решения множества раз-

личных задач. К сожалению, они также имеют серьезные недостатки. Они нуждаются в большом количестве параметров, что делает их очень восприимчивыми к переобучению. Их становится трудно обучать, когда они имеют более одного или двух скрытых слоев. Во многих случаях вы можете получить лучший результат, используя менее общую модель, которая соответствует специфическим особенностям вашей проблемы.

Большая часть содержания этой книги состоит из обсуждения конкретных разновидностей моделей, наиболее полезных в науках о жизни. Они могут подождать до следующих глав. Но для полноты этого введения мы должны сейчас обсудить два очень важных класса моделей, широко применяемых во многих областях. Их называют сверточными нейронными сетями и рекуррентными нейронными сетями.

Сверточные нейронные сети

Сверточные нейронные сети (convolutional neural networks, CNN) были одной из самых первых разновидностей глубоких моделей, которые получили широкое распространение. Они были разработаны для использования в обработке изображений и компьютерном зрении и остаются отличным выбором для многих задач, основанных на непрерывных данных, взятых по прямоугольной сетке: аудиосигналы (одномерные), изображения (двумерные), объемные данные МРТ (трехмерные) и т. д.

Они также представляют собой класс моделей, которые действительно оправдывают термин «нейронная сеть». Устройство CNN основано на исследованиях работы зрительной коры кошачьих. Кошки вообще сыграли центральную роль в глубоком обучении на заре этой отрасли. Исследования, проведенные с 1950-х по 1980-е годы, показали, что изображение обрабатывается через ряд слоев. Каждый нейрон в первом слое получает входной сигнал из небольшой области поля зрения – своего *рецепторного поля* (receptive field). Различные нейроны специализируются на обнаружении определенных локальных паттернов или признаков, таких как вертикальные или горизонтальные линии. Клетки во втором слое получают входные данные от кластеров ячеек первого слоя, объединяя их сигналы для обнаружения более сложных паттернов в большем рецепторном поле. Каждый слой можно рассматривать как новое представление исходного изображения, описанное в терминах более крупных и более абстрактных паттернов, чем в предыдущем слое.

CNN воспроизводят эту структуру, пропуская входное изображение через ряд слоев. В этом смысле они похожи на MLP, но строение и связь слоев сильно отличаются. MLP используют *полностью связанные слои*, где каждый элемент выходного вектора зависит от каждого элемента входного вектора. CNN используют *сверточные слои*, которые применяют преимущества пространственной локализации. Каждый выходной элемент соответствует небольшой области изображения и зависит только от входных значений в этой области. Это значительно уменьшает количество параметров, определяющих каждый слой. В сущности, предполагается, что большинство элементов весовой матрицы M_i равно нулю, поскольку каждый выходной элемент зависит только от небольшого количества входных элементов.

Сверточные слои идут еще дальше и предполагают, что параметры одинаковы для *каждой локальной области изображения*. Если слой использует некий набор параметров для обнаружения горизонтальных линий в одном месте изображения, он использует точно такие же параметры для обнаружения горизонтальных

линий повсюду в изображении. Это делает количество параметров для слоя независимым от размера изображения. Все, что ему нужно выучить, – это одно *сверточное ядро*, которое определяет, как выходные признаки вычисляются из любой локальной области изображения. Эта локальная область зачастую очень маленькая, например 5×5 пикселей. В таком случае количество изучаемых параметров только в 25 раз превышает количество выходных признаков для каждого региона. Это ничтожно мало по сравнению с количеством в полностью подключенном слое, что делает CNN намного проще в обучении и гораздо устойчивее к переобучению, чем MLP.

Рекуррентные нейронные сети

Рекуррентные нейронные сети (recurrent neural networks, RNN) немного отличаются от сверточных сетей. Обычно они используются для обработки данных в виде последовательности элементов: слов в текстовом документе, оснований в молекуле ДНК и т. п. Элементы последовательности подаются на ввод сети по одному. А затем сеть делает нечто необычное. На следующем шаге вывод из каждого слоя возвращается на свой собственный ввод! Это позволяет RNN иметь своего рода память. Когда элемент последовательности подается в сеть, ввод каждого слоя зависит как от этого элемента, так и от всех предыдущих элементов, которые были введены раньше.

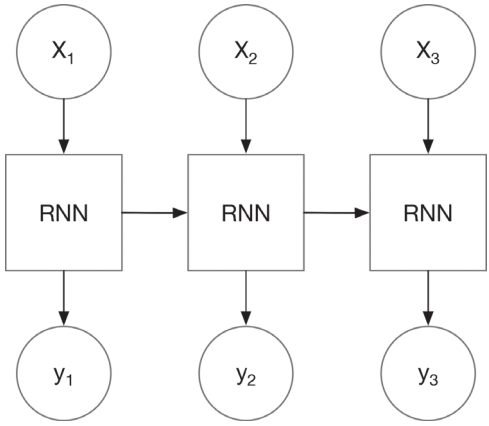


Рис. 2.4 ❖ Рекуррентная нейронная сеть.
Вывод (y_1, y_2, ...) зависит как от элемента (x_1, x_2, ...) на вводе,
так и от собственного вывода RNN во время предыдущего шага

Таким образом, ввод для рекуррентного слоя состоит из двух частей: обычного ввода (то есть вывода из предыдущего слоя в сети) и повторного ввода (который равен его собственному выводу на предыдущем шаге). Далее происходит расчет нового результата на основе этих входных данных. В принципе, вы можете использовать полностью связанный слой, но на практике это обычно не очень хорошо работает. Исследователи разработали другие типы слоев, которые работают в RNN намного лучше. Наиболее популярны *управляемый рекуррентный блок* (gated-recurrent unit, GRU) и *долгая краткосрочная память* (long short-term memory, LSTM).

Не беспокойтесь о деталях, просто запомните названия. При создании рекуррентной нейросети вы обычно должны использовать один из этих двух типов слоев.

Наличие памяти принципиально отличает RNN от других моделей, которые мы обсуждали. Используя CNN или MLP, вы просто подаете данные на вход сети и получаете значения на выходе. Вывод полностью определяется текущим вводом. RNN работает иначе. Модель имеет собственное «внутреннее состояние» – выводы всех слоев на самом последнем шаге. Каждый раз, когда вы вводите в модель новое значение, вывод зависит не только от входного значения, но и от внутреннего состояния. Аналогично, внутреннее состояние изменяется с каждым новым входным значением. Это свойство делает рекуррентные нейросети очень мощными и позволяет использовать их для множества различных приложений.

Дополнительное чтение

В этой главе дано только краткое введение в глубокое обучение. Его достаточно, чтобы прочитать и понять остальную часть этой книги, но если вы планируете серьезную работу в этой области, вам необходимо получить гораздо более основательную подготовку. К счастью, в интернете есть много отличных ресурсов по теме глубокого обучения. Предлагаем вам ознакомиться с некоторыми полезными материалами.

- Бесплатная онлайн-книга[1] «Нейронные сети и глубокое обучение» Майкла Нильсена охватывает примерно тот же материал, что и в этой главе, но углубляется в подробности по каждой теме. Если вам нужны прочные практические знания основ глубокого обучения, достаточные для того, чтобы использовать их в своей работе, это отличное место для начала.
- Еще одна онлайн-книга[2] «Глубокое обучение» Яна Гудфеллоу, Йошуа Бенджио и Аарона Курвилла – это более продвинутое введение, написанное известными ведущими исследователями в данной области. Они ожидают, что читатель имеет подготовку на уровне аспирантуры в области информатики и углубится в математические теории, стоящие за этим предметом. Вы можете легко использовать глубокие модели, не вникая в теорию, но если вы хотите провести оригинальные исследования именно в области глубокого обучения, а не просто использовать глубокие модели в качестве инструмента для решения проблем в других областях, то эта книга станет для вас фантастическим источником знаний.
- «TensorFlow для глубокого обучения» Бхарата Рамсундара и Резы Босаг Заде[3] представляет собой введение в глубокое обучение для практиков, формирующее интуитивное понимание основных концепций, не вдаваясь слишком глубоко в математические основы глубоких моделей. Книга может быть полезным справочным материалом для исследователей, интересующихся прикладными аспектами глубокого обучения.

---

[1] *Nilsen M.* Neural Networks and Deep Learning // **http://neuralnetworksanddeeplearning.com**.

[2] *Goodfellow I., Bengio Y., Courville A.* Deep Learning // **http://www.deeplearningbook.org**.

[3] *Рамсундар Бхарат, Реза Босаг Заде.* TensorFlow для глубокого обучения. СПб.: БХВ-Петербург, 2019. ISBN 978-5-9775-4014-8.

Глава 3

Машинное обучение с DeepChem

В этой главе дается краткое введение в машинное обучение с DeepChem – библиотекой, специально созданной на основе платформы TensorFlow, чтобы облегчить применение глубокого обучения в науках о жизни. DeepChem предоставляет широкий выбор моделей, алгоритмов и наборов данных, которые подходят для разработки приложений в области наук о жизни. В оставшейся части этой книги мы будем использовать DeepChem для проведения наших тематических исследований.

Почему бы просто не использовать Keras, TensorFlow или PyTorch?

Разработчиков DeepChem постоянно спрашивают, почему бы просто не использовать Keras, TensorFlow или PyTorch? Короткий ответ заключается в том, что разработчики этих пакетов концентрируют свое внимание на поддержке определенных сценариев применения, полезных для большинства пользователей. Например, существует обширная поддержка обработки изображений, обработки текста и анализа речи. Но, как правило, в этих библиотеках не предусмотрена аналогичная поддержка для обработки молекул, наборов генетических данных или наборов данных микроскопии. Назначение DeepChem – предоставить первоклассную поддержку в виде библиотеки именно таким приложениям. Это означает создание пользовательских примитивов глубокого обучения, поддержку необходимых типов файлов, а также обширные учебные пособия и документацию.

DeepChem спроектирован так, чтобы хорошо включаться в экосистему TensorFlow, поэтому вы сможете сочетать и смешивать код DeepChem с остальным кодом вашего приложения TensorFlow.

Далее мы будем предполагать, что на вашем компьютере установлена библиотека DeepChem и вы готовы запустить примеры кода. Если это не так, не беспокойтесь. Просто зайдите на сайт DeepChem по адресу **https://deepchem.io** и следуйте инструкциям по установке для вашей системы.

Совместимость с Windows

В настоящее время DeepChem не поддерживает установку в Windows. Мы рекомендуем вам ознакомиться с примерами из этой книги на рабочей станции Mac или Linux. Мы слышали от наших пользователей, что DeepChem работает на подсистеме Windows для Linux (Windows Subsystem for Linux, WSL) в современных дистрибутивах Windows[1].

[1] Подсистема WSL предназначена для запуска исполняемых файлов Linux в формате ELF на компьютерах с ОС Windows 10 (только 64-разрядная версия, начиная с релиза 1607) или Windows Server 2019. – *Прим. перев*.

> Если вы не можете получить доступ к компьютеру Mac или Linux или работать с WSL, мы хотим помочь вам решить проблему совместимости DeepChem с Windows. Пожалуйста, свяжитесь с авторами по поводу затруднений, с которыми вы столкнулись, и мы постараемся устранить их. Мы надеемся снять ограничение совместимости к выходу следующих изданий книги.

Наборы данных DeepChem

DeepChem использует базовую абстракцию объекта `Dataset` для переноса данных, предназначенных для машинного обучения. `Dataset` содержит информацию о наборе выборок: входные векторы **x**, выходные векторы **y** и, возможно, другая информация, такая как описание того, что представляет каждая выборка. Существуют подклассы объекта `Dataset`, соответствующие различным способам хранения данных. В частности, мы будем использовать объект `NumpyDataset`, служащий удобной оберткой для массивов NumPy. В этом разделе мы рассмотрим простой пример использования `NumpyDataset`. Начнем с обычного импорта:

```
import deepchem as dc
import numpy as np
```

Далее создадим пару простых массивов NumPy:

```
x = np.random.random((4, 5))
y = np.random.random((4, 1))
```

Этот набор данных будет содержать четыре выборки. Массив `x` имеет пять элементов (*признаков*) для каждой выборки, а массив `y` имеет один элемент для каждой выборки. Посмотрите, как выглядят реальные массивы выборок. Имейте в виду, что когда вы запускаете этот код на своем компьютере, вы должны увидеть другие числа, так как ваше *случайное начальное число* (random seed) будет другим.

```
In: x
Out:
array([[0.960767 , 0.31300931, 0.23342295, 0.59850938, 0.30457302],
       [0.48891533, 0.69610528, 0.02846666, 0.20008034, 0.94781389],
       [0.17353084, 0.95867152, 0.73392433, 0.47493093, 0.4970179 ],
       [0.15392434, 0.95759308, 0.72501478, 0.38191593, 0.16335888]])

In: y
Out:
array([[0.00631553],
       [0.69677301],
       [0.16545319],
       [0.04906014]])
```

Теперь обернем эти массивы в объект `NumpyDataset`:

```
dataset = dc.data.NumpyDataset(x, y)
```

Мы можем «развернуть» объект `dataset`, чтобы получить исходные массивы, хранящиеся внутри него:

```
In: print(dataset.X)
[[0.960767 0.31300931 0.23342295 0.59850938 0.30457302]
```

```
 [0.48891533 0.69610528 0.02846666 0.20008034 0.94781389]
 [0.17353084 0.95867152 0.73392433 0.47493093 0.4970179 ]
 [0.15392434 0.95759308 0.72501478 0.38191593 0.16335888]]

In: print(dataset.y)
[[0.00631553]
 [0.69677301]
 [0.16545319]
 [0.04906014]]
```

Можно убедиться, что извлеченные массивы идентичны исходным массивам x и y:

```
In [23]: np.array_equal(x, dataset.X)
Out[23]: True

In [24]: np.array_equal(y, dataset.y)
Out[24]: True
```

Другие типы наборов данных

DeepChem поддерживает другие типы наборов данных, как упоминалось ранее. Эти типы объектов наборов данных в первую очередь становятся полезными при работе с большими наборами данных, которые невозможно полностью сохранить в памяти компьютера. DeepChem также может использовать утилиты загрузки набора данных tf.data от TensorFlow. Мы рассмотрим эти более продвинутые функции библиотеки по мере их необходимости.

Обучение модели для предсказания токсичности молекул

В этом разделе мы узнаем, как использовать DeepChem для обучения модели для прогнозирования токсичности молекул. В следующей главе мы гораздо глубже объясним, как работает предсказание токсичности молекул, но в этом разделе мы рассмотрим его в качестве примера использования моделей DeepChem для решения задач машинного обучения. Давайте начнем с пары необходимого импорта.

```
import numpy as np
import deepchem as dc
```

Следующим шагом является загрузка соответствующих наборов данных о токсичности для обучения модели машинного обучения. DeepChem поддерживает модуль `dc.molnet` (сокращение от MoleculeNet), который содержит ряд предварительно обработанных наборов данных для использования в экспериментах по машинному обучению. В частности, мы будем использовать функцию `dc.molnet.load_tox21()`, которая будет загружать и обрабатывать для нас набор данных о токсичности Tox21. Когда вы запускаете эти команды в первый раз, DeepChem будет обрабатывать набор данных локально на вашем компьютере. Вы должны увидеть вывод на экран, похожий на показанный ниже:

```
In : tox21_tasks, tox21_datasets, transformers = dc.molnet.load_tox21()
Loading raw samples now.
shard_size: 8192
About to start loading CSV from /tmp/tox21.csv.gz
Loading shard 1 of size 8192.
Featurizing sample 0
```

```
Featurizing sample 1000
Featurizing sample 2000
Featurizing sample 3000
Featurizing sample 4000
Featurizing sample 5000
Featurizing sample 6000
Featurizing sample 7000
TIMING: featurizing shard 0 took 15.671 s
TIMING: dataset construction took 16.277 s
Loading dataset from disk.
TIMING: dataset construction took 1.344 s
Loading dataset from disk.
TIMING: dataset construction took 1.165 s
Loading dataset from disk.
TIMING: dataset construction took 0.779 s
Loading dataset from disk.
TIMING: dataset construction
```

Получение *признакового описания* (feature vector), называемое *фичеризацией* (featurization), – это преобразование набора данных, содержащего информацию о молекулах, в матрицы и векторы для использования в анализе машинного обучения. Мы будем более подробно исследовать процесс фичеризации в последующих главах, а сейчас бегло взглянем на данные, которые мы обработали. Функция `dc.molnet.load_tox21()` возвращает несколько типов выходных данных: `tox21_tasks`, `tox21_datasets` и `transformers`. Давайте кратко рассмотрим каждый из них.

```
In : tox21_tasks
Out:
['NR-AR',
'NR-AR-LBD',
'NR-AhR',
'NR-Aromatase',
'NR-ER',
'NR-ER-LBD',
'NR-PPAR-gamma',
'SR-ARE',
'SR-ATAD5',
'SR-HSE',
'SR-MMP',
'SR-p53']
In : len(tox21_tasks)
Out: 12
```

Каждое из этих 12 *заданий* (tasks) здесь соответствует определенному биологическому эксперименту. В нашем случае каждая из этих задач предназначена для *ферментативного анализа* (enzymatic assay), то есть проверки, связываются ли молекулы в наборе данных Tox21 с рассматриваемой *биологической мишенью* (biological target). Термины «NR-AR» и им подобные из приведенного выше списка обозначают мишени. В данном случае каждая мишень представляет собой определенный фермент, который, как полагают, связан с токсическими реакциями на молекулы потенциального лекарства.

Насколько хорошо мне следует знать биологию?

У специалистов по информатике и инженеров, мало знакомых с науками о жизни, обилие биологических терминов может вызвать головокружение. Тем не менее не обязательно иметь глубокое понимание биологии, чтобы начать разбираться в науках о жизни. Если вы специалист в области информатики, попробуйте понять биологические системы с точки зрения компьютерных аналогов. Представьте, что клетки или животные – это сложные устаревшие базы кодов, которые вы не можете контролировать. Будучи инженером, вы обладаете набором экспериментальных измерений этих систем (анализов), с помощью которых можете получить некоторое представление об устройстве системы. Машинное обучение является чрезвычайно мощным инструментом для понимания биологических систем, поскольку алгоритмы обучения способны автоматически находить полезные корреляции. Это позволяет даже новичкам в биологии иногда находить глубокие научные идеи.

В оставшейся части этой книги мы кратко обсудим основы биологии. Пусть эти заметки послужат вам отправной точкой в изучении обширной биологической литературы. Даже открытые общественные ресурсы, такие как Википедия, содержат огромное количество полезной информации в области биологии.

Далее давайте рассмотрим `tox21_datasets`. Использование множественного числа является подсказкой, что это поле на самом деле является кортежем, содержащим несколько объектов `dc.data.Dataset`:

```
In : tox21_datasets
Out:
(<deepchem.data.datasets.DiskDataset at 0x7f9804d6c390>,
<deepchem.data.datasets.DiskDataset at 0x7f9804d6c780>,
<deepchem.data.datasets.DiskDataset at 0x7f9804c5a518>)
```

В данном случае эти наборы данных соответствуют обучающим, проверочным и оценочным наборам, о которых вы узнали в предыдущей главе. Вы можете заметить, что это объекты типа `DiskDataset`; модуль `dc.molnet` кеширует эти наборы данных на вашем диске, так что вам не нужно выполнять повторную фичеризацию набора данных Tox21. Давайте корректно разделим эти наборы данных.

```
train_dataset, valid_dataset, test_dataset = tox21_datasets
```

Работу с новыми наборами данных очень полезно начать с изучения вида даных. Для этого взгляните на атрибут `shape`.

```
In : train_dataset.X.shape
Out: (6264, 1024)

In : valid_dataset.X.shape
Out: (783, 1024)

In : test_dataset.X.shape
Out: (784, 1024)
```

Набор `train_dataset` содержит в общей сложности 6264 выборки, каждая из которых имеет связанный вектор признаков длиной 1024. Аналогично, `valid_dataset` и `test_dataset` содержат соответственно 783 и 784 выборки. Давайте теперь кратко рассмотрим векторы **y** для этих наборов данных.

```
In : np.shape(train_dataset.y)
Out: (6264, 12)
```

```
In : np.shape(valid_dataset.y)
Out: (783, 12)

In : np.shape(test_dataset.y)
Out: (784, 12)
```

Для каждой выборки имеется 12 точек данных, также называемых *метками* (labels). Они соответствуют 12 заданиям, которые мы обсуждали выше. В этом конкретном наборе данных выборки соответствуют молекулам, задания соответствуют биохимическим анализам, и каждая метка является результатом конкретного ферментативного анализа конкретной молекулы. Это материал, на котором мы хотим научить нашу модель прогнозировать токсичность новых молекул.

Но есть одна сложность. Реальный экспериментальный набор данных для Tox21 не содержит результаты проверки *каждой* молекулы в *каждом* биологическом эксперименте. Это означает, что некоторые из меток являются бессмысленными заполнителями. У нас просто нет никаких данных для некоторых свойств некоторых молекул, поэтому мы должны игнорировать эти элементы массивов при обучении и проверке модели.

Как мы можем узнать, какие метки на самом деле имеют смысл? Мы можем проверить поле `w` набора данных, в котором записаны *веса* (weights) меток. Всякий раз, когда мы вычисляем функцию потерь для модели, мы умножаем значения на `w` перед суммированием по задачам и выборкам. Веса могут быть использованы для нескольких целей, одной из которых является пометка отсутствующих данных. Если метка имеет вес 0, эта метка не влияет на потерю и игнорируется во время обучения. Давайте немного покопаемся в данных, чтобы узнать, сколько меток фактически было измерено в наших наборах.

```
In : train_dataset.w.shape
Out: (6264, 12)

In : np.count_nonzero(train_dataset.w)
Out: 62166

In : np.count_nonzero(train_dataset.w == 0)
Out: 13002
```

Таким образом, из $6264 \times 12 = 75\,168$ элементов в массиве меток фактически были измерены только 62 166. Остальные 13 002 метки соответствуют отсутствующим измерениям и должны игнорироваться. Вы можете спросить, зачем мы до сих пор храним такие записи? Ответ прост: в основном для удобства. Работать с массивами неправильной формы гораздо труднее, чем с обычными матрицами со связанным набором весов.

Обработка наборов данных – сложная задача

Важно отметить, что очистка и обработка набора данных для использования в науках о жизни может оказаться чрезвычайно сложной задачей. Многие необработанные наборы данных будут содержать систематические ошибки. Если рассматриваемый набор данных был создан на основе эксперимента, проведенного внешней контрактной организацией (contract research organization, CRO), вполне возможно, что набор данных будет систематически неверным. По этой причине многие организации, занимающиеся науками о жизни, имеют собственных ученых, чья работа заключается в проверке и очистке таких наборов данных. В целом если ваш алгоритм машинного обучения не работает для задачи в области наук о жизни, вполне вероятно, что основная причина неполадок связана не с алгоритмом, а с систематическими ошибками в источнике данных, которые вы используете.

Теперь давайте рассмотрим набор `transformers`, конечный результат, который был возвращен `load_tox21()`. *Преобразователь* – это объект, который каким-то образом модифицирует набор данных. DeepChem предоставляет множество преобразователей, которые выполняют полезные преобразования данных. Процедуры загрузки данных, применяемые в MoleculeNet, всегда возвращают список преобразователей, которые были применены к данным, так как они могут пригодиться позже, чтобы «отменить» преобразование. Давайте посмотрим, какое преобразование применяется в нашем случае:

```
In : transformers
Out: [<deepchem.trans.transformers.BalancingTransformer at 0x7f99dd73c6d8>]
```

Итак, наши данные были преобразованы с помощью `BalancingTransformer`. Этот класс используется для исправления несбалансированных данных. В случае Tox21 большинство молекул не связывается с большинством мишеней. Фактически более 90 % меток равны 0. Это означает, что модель могла бы достичь точности более 90 %, просто прогнозируя 0 независимо от того, какой ввод был дан. К сожалению, такая модель абсолютно бесполезна! Несбалансированные данные, где для одних классов гораздо больше обучающих выборок, чем для других, являются распространенной проблемой в задачах классификации.

К счастью, есть простое решение: скорректировать матрицу весов набора данных. `BalancingTransformer` корректирует веса для отдельных точек данных так, чтобы общий вес, назначенный каждому классу, был одинаковым. Таким образом, функция потерь не имеет систематического предпочтения для какого-либо одного класса. Потери можно уменьшить, только научившись правильно различать классы.

Теперь, когда мы изучили наборы данных Tox21, давайте начнем осваивать обучение моделей на этих наборах данных. Подмодуль DeepChem `dc.models` содержит множество различных моделей, связанных с науками о жизни. Все эти модели наследуются от родительского класса `dc.models.Model`. Этот родительский класс предназначен для предоставления общего API в соответствии с соглашениями Python. Если вы раньше использовали другие пакеты машинного обучения Python, то должны заметить, что многие методы `dc.models.Model` выглядят довольно знакомо.

В этой главе мы не будем углубляться в детали того, как создаются модели, а просто приведем пример создания экземпляра стандартной модели DeepChem, – `dc.models.MultitaskClassifier`. Наша модель строит полностью подключенную сеть (MLP), отображающую входные признаки на несколько выходных прогнозов. Это делает модель полезной для *многозадачных* (multitask) проблем, когда для каждой выборки есть несколько меток. Модель хорошо подходит для наборов данных Tox21, поскольку у нас есть в общей сложности 12 различных анализов, результаты которых мы хотим прогнозировать одновременно. Давайте посмотрим, как мы можем построить `MultitaskClassifier` в DeepChem:

```
model = dc.models.MultitaskClassifier(n_tasks=12,
n_features=1024,
layer_sizes=[1000])
```

Здесь мы видим несколько параметров. Давайте кратко поясним их назначение. `n_tasks` – это количество задач, а `n_features` – количество входных признаков для каждой выборки. Как мы видели выше, набор данных Tox21 имеет 12 задач

и 1024 признака для каждой выборки. `layer_sizes` – это список, который устанавливает количество полностью подключенных скрытых слоев в сети и ширину каждого из них. В данном случае мы указываем, что существует только один скрытый слой шириной 1000.

Теперь, когда модель построена, необходимо обучить ее на наборах данных Tox21. Каждый объект `Model` имеет метод `fit()`, который подгоняет модель к данным, содержащимся в объекте набора данных. Подгонка нашего объекта `MultitaskClassifier` выполняется простым вызовом метода.

```
model.fit(train_dataset, nb_epoch=10)
```

Обратите внимание на новый параметр `nb_epoch=10`, говорящий о том, что будет проведено 10 *эпох* (epoch) обучения градиентным спуском. Эпоха относится к одному полному проходу через все выборки в наборе данных. Чтобы обучить модель, вы разделяете обучающий набор на пакеты и выполняете один шаг градиентного спуска для каждого пакета. В идеальном мире вы бы получили хорошо оптимизированную модель, прежде чем закончатся данные. На практике данных для обучения обычно не хватает, поэтому у вас заканчиваются данные до того, как модель полностью обучена. Вам приходится повторно использовать данные, делая дополнительные проходы через набор. Это позволяет обучать модели с меньшим количеством данных, но чем больше эпох вы используете, тем больше вероятность того, что вы получите переобученную модель.

Теперь оценим качество обученной модели. Чтобы оценить, насколько хорошо работает модель, необходимо указать *метрику* – критерий оценки. Класс DeepChem `dc.metrics.Metric` предоставляет общий способ задания метрик для моделей. Для наборов данных Tox21 полезным показателем является ROC-AUC, поэтому давайте проведем анализ, используя этот показатель. Однако у нас есть несколько задач Tox21. Для какой из них мы рассчитываем ROC-AUC? Хорошим решением будет вычисление среднего значения ROC-AUC для всех задач. К счастью, это легко сделать:

```
metric = dc.metrics.Metric(dc.metrics.roc_auc_score, np.mean)
```

ROC-AUC

Мы хотим классифицировать молекулы как токсичные или нетоксичные, но модель выдает непрерывные числа, а не дискретные прогнозы. На практике вы выбираете пороговое значение и прогнозируете, что молекула токсична всякий раз, когда выходной сигнал превышает пороговое значение. Слишком низкий порог даст много ложных срабатываний (предсказаний, что безопасная молекула якобы токсична). Слишком высокий порог даст меньше ложных срабатываний, но больше ложных отрицательных результатов (неверный прогноз, что токсичная молекула безопасна).

Кривая рабочих характеристик приемника (receiver operating characteristic, ROC) является удобным способом визуализации этого компромисса. Вы пробуете много разных пороговых значений, а затем строите кривую по двум параметрам – доля истинно положительных прогнозов и доля ложноположительных прогнозов – для каждого порогового значения. Пример показан на рис. 3.1.

ROC-AUC – это *общая площадь под кривой* ROC (Area Under Curve). Если существует пороговое значение, для которого *каждая* выборка классифицирована правильно, ROC-AUC равен 1. С другой стороны, если модель выдает полностью случайные значения, не связанные с истинными классами, ROC-AUC составляет 0,5. Это свойство делает ROC-AUC полезным параметром для оценки качества работы классификатора.

Поскольку мы указали значение `np.mean`, будет вычисляться среднее значение ROC-AUC по всем задачам. Модели DeepChem поддерживают функцию оценки `Model.evaluate()`, которая оценивает производительность модели по заданному набору данных и метрике.

```
train_scores = model.evaluate(train_dataset, [metric], transformers)
est_scores = model.evaluate(test_dataset, [metric], transformers)
```

Теперь, когда мы подсчитали оценки, давайте посмотрим, что получилось:

```
In : print(train_scores)
...: print(test_scores)
...:
{'mean-roc_auc_score': 0.9659541853946179}
{'mean-roc_auc_score': 0.7915464001982299}
```

Обратите внимание, что оценка на обучающем наборе (0,96) намного лучше, чем оценка на проверочном наборе (0,79). Это показывает, что модель была переобучена. На самом деле нас интересует результат для проверочного набора, и он не самый лучший для этого набора данных. На момент написания статьи лучшие результаты по усредненной оценке ROC-AUC для набора данных Tox21 были чуть ниже 0,9. В то же время эти результаты совсем не плохи для неотлаженной системы. Полная кривая ROC для одной из 12 задач показана на рис. 3.1.

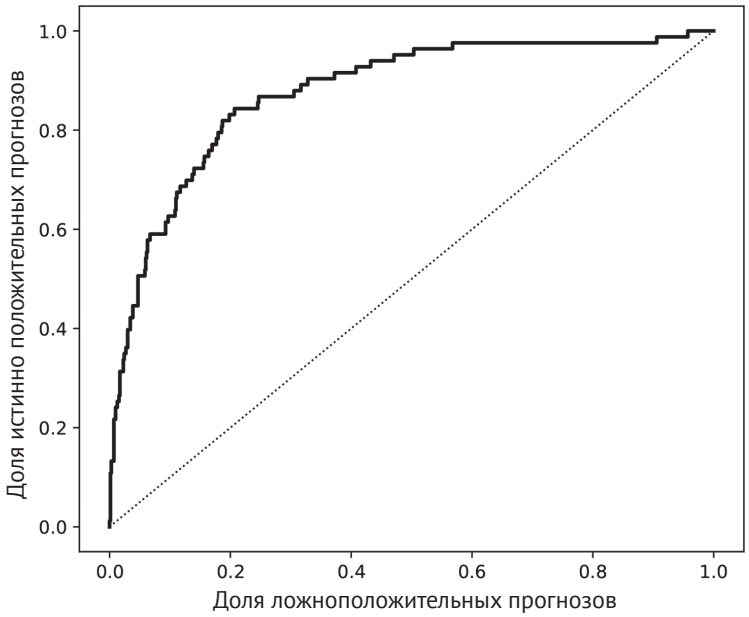


Рис. 3.1 ❖ Кривая ROC для одной из 12 задач. Пунктирная диагональная линия показывает, какой будет кривая для модели, дающей абсолютно случайный прогноз. Фактическая кривая находится значительно выше диагонали, показывая, что модель дает прогноз намного лучше, чем случайные предположения

Пример: обучение модели MNIST

В предыдущем разделе мы рассмотрели основы обучения модели машинного обучения с помощью DeepChem. Однако мы использовали готовый модельный класс `dc.models.MultitaskClassifier`. Иногда бывает необходимо вместо использования предварительно настроенной архитектуры создать новую архитектуру глубокого обучения. В этом разделе мы обсудим, как обучить сверточную нейронную сеть на наборе данных для распознавания цифр MNIST. Вместо того чтобы использовать готовую архитектуру, как в предыдущем примере, на этот раз мы сами определим полную архитектуру глубокого обучения и воспользуемся классом `dc.models.TensorFraph`, предоставляющим фреймворк для построения глубокой архитектуры в DeepChem.

Когда уместно применять «консервированные» модели?

В этом разделе для работы с набором MNIST мы собираемся создать пользовательскую архитектуру. В предыдущем примере мы использовали заранее подготовленную, или, как иногда говорят, «консервированную» (canned) архитектуру. Когда имеет смысл применять готовую архитектуру? Если у вас есть хорошо отлаженная постоянная архитектура для известной проблемы, вероятно, имеет смысл использовать именно ее. Но если вы работаете с новым набором данных, вам может потребоваться собственная архитектура. Важно уметь применять как готовую, так и пользовательскую архитектуру, поэтому мы привели в этой главе примеры использования каждой из них.

Набор данных распознавания цифр MNIST

Набор данных распознавания изображений MNIST (рис. 3.2) предназначен для построения модели машинного обучения, различающей рукописные цифры. Задача состоит в том, чтобы классифицировать цифры от 0 до 9, исходя из черно-белых изображений размером 28×28 пикселей. Набор данных содержит 60 000 обучающих выборок и проверочный набор из 10 000 выборок.

Рис. 3.2 ❖ Образцы взяты из набора данных распознавания рукописных цифр MNIST[1]

[1] **https://github.com/mnielsen/rmnist/blob/master/data/rmnist_10.png**.

С точки зрения машинного обучения набор данных MNIST не особенно сложен. Десятилетия исследований позволили создать современные алгоритмы, которые достигают почти 100%-ной точности классификации для этого набора данных. В результате набор данных MNIST больше не подходит для исследовательской работы, но является хорошим инструментом для педагогических целей.

Разве DeepChem подходит не только для наук о жизни?

Как мы уже говорили, вполне возможно создавать приложения для наук о жизни с использованием других пакетов глубокого обучения. Точно так же вполне можно создать общие системы машинного обучения, используя DeepChem. Хотя создание системы рекомендаций фильмов в DeepChem может оказаться более сложным, чем со специальными инструментами, это было бы вполне осуществимо. Было проведено множество исследований, посвященных использованию рекомендательных алгоритмов для предсказания молекулярного связывания. Архитектуры машинного обучения, разработанные для одной области, поддаются переносу на другие области, поэтому важно проявлять гибкость, столь необходимую для исследовательской работы.

Сверточная архитектура для набора MNIST

Для создания нестандартных архитектур глубокого обучения DeepChem использует класс `TensorGraph`. В этом разделе мы рассмотрим код, необходимый для построения сверточной архитектуры, показанной на рис. 3.3. Он начинается с двух сверточных слоев для определения локальных признаков изображения. За ними следуют два полностью связанных слоя, предсказывающих цифру, исходя из этих локальных признаков.

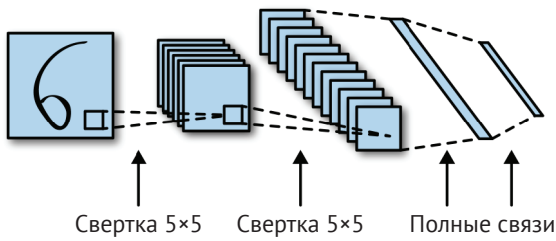


Рис. 3.3 ❖ Иллюстрация архитектуры, которую мы построим в этом разделе для обработки набора MNIST

Начнем со скачивания файлов данных MNIST на свой компьютер:

```
mkdir MNIST_data
cd MNIST_data
wget http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
wget http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
wget http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
wget http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
cd ..
```

Выполнив эти команды, вы скачали набор данных MNIST и сохранили его на локальном компьютере. Давайте теперь подключим эти файлы данных.

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Теперь преобразуем эти данные в формат, подходящий для анализа при помощи DeepChem. Давайте начнем с того, что сделаем необходимый импорт:

```
import deepchem as dc
import tensorflow as tf
import deepchem.models.tensorgraph.layers as layers
```

Подмодуль `deepchem.models.tensorgraph.layers` содержит коллекцию так называемых *слоев*. Эти слои служат строительными блоками архитектур и могут быть объединены для создания новых архитектур глубокого обучения. Скоро мы покажем, как использовать объекты слоя. Далее мы создаем объекты `NumpyDataset`, которые обертывают наборы обучающих и проверочных данных MNIST:

```
train_dataset = dc.data.NumpyDataset(mnist.train.images, mnist.train.labels)
test_dataset = dc.data.NumpyDataset(mnist.test.images, mnist.test.labels)
```

Обратите внимание, что хотя изначально мы не задавали проверочный набор, функция `input_data` из TensorFlow сама позаботится о выделении правильного проверочного набора данных для последующего использования. Имея наборы данных для обучения и проверки, мы можем сосредоточиться на определении архитектуры сверточной сети MNIST.

Ключевая идея заключается в том, что объекты слоев можно комбинировать, составляя из них новые модели. Как мы обсуждали в предыдущей главе, каждый слой получает входные данные от предыдущих слоев и вычисляет выходные данные, которые могут быть переданы последующим слоям. В самом начале модели расположены входные слои, которые принимают объекты и метки. На другом конце находятся выходные слои, которые возвращают результаты выполненных вычислений. В этом примере мы составим последовательность слоев, чтобы построить сверточную сеть обработки изображений. Сначала мы определим новый объект `TensorGraph`:

```
model = dc.models.TensorGraph(model_dir='mnist')
```

Опция `model_dir` указывает каталог, в котором должны быть сохранены параметры модели. Вы можете опустить этот параметр, как мы делали в предыдущем примере, но тогда модель не будет сохранена. Как только интерпретатор Python завершит выполнение программы, все ваши усилия по обучению модели пропадут напрасно! Указав каталог, вы можете позже перезагрузить модель и сделать с ней новые прогнозы.

Обратите внимание: поскольку `TensorGraph` унаследован от `Model`, этот объект является экземпляром класса `dc.models.Model` и поддерживает те же функции `fit()` и `evaluate()`, которые мы встречали ранее.

```
In : isinstance(model, dc.models.Model)
Out: True
```

Мы еще ничего не добавили в модель, поэтому наша модель вряд ли будет очень интересной. Давайте начнем с добавления некоторых входных данных для признаков и меток с помощью классов `Feature` и `Label`.

```
feature = layers.Feature(shape=(None, 784))
label = layers.Label(shape=(None, 10))
```

MNIST содержит изображения размером 28×28. При выравнивании эти матрицы образуют векторы длины 784. Размерность меток равна десяти, поскольку существует десять возможных значений цифр, а для кодирования вектора применяется *прямой унитарный код* (one-hot encode). Обратите внимание, что в качестве размерности входных данных используется `None`. В системах, основанных на TensorFlow, значение `None` часто означает способность данного слоя принимать входные данные, имеющие любой размер в данном измерении. Иными словами, наш объект `feature` способен с равным успехом принимать входные данные вида (20, 784) и (97, 784). В данном случае первый размер соответствует размеру пакета, поэтому наша модель сможет принимать пакеты с любым количеством выборок.

Прямое унитарное кодирование

Набор данных MNIST является *категорийным*, то есть объекты принадлежат к одной из категорий, входящих в конечный список. В данном случае эти категории представляют собой цифры от 0 до 9. Как мы можем ввести эти категории в систему машинного обучения? Одним из очевидных ответов было бы просто ввести одну переменную, которая принимает значения от 0 до 9. Однако по ряду технических причин такое кодирование нередко оказывается неэффективным. Часто применяемой альтернативой является *прямое унитарное кодирование*, иногда называемое *одинарным горячим кодированием* (one-hot encode). Каждая метка для MNIST представляет собой вектор длиной 10, в котором только для одного элемента задано значение 1, а для всех остальных – ноль. Если единица находится в нулевом индексе, то метка соответствует цифре 0. Если единица находится в девятом индексе, то метка соответствует цифре 9.

Чтобы применить сверточные слои к нашему входу, нам нужно преобразовать наши плоские векторы признаков в матрицы (28, 28). Для этого мы будем использовать слой `Reshape`.

```
make_image = layers.Reshape(shape=(None, 28, 28), in_layers=feature)
```

Здесь снова значение `None` указывает на возможность обработки пакетов произвольного размера. Обратите внимание на аргумент `in_layers=feature`. Он указывает на то, что слой Reshape принимает в качестве входных данных наш предыдущий слой Feature. Теперь, когда мы успешно изменили входные данные, мы можем передать их сверточным слоям.

```
conv2d_1 = layers.Conv2D(num_outputs=32, activation_fn=tf.nn.relu, in_layers=make_image)
conv2d_2 = layers.Conv2D(num_outputs=64, activation_fn=tf.nn.relu, in_layers=conv2d_1)
```

Здесь класс `Conv2D` применяет двумерную свертку к каждой выборке своего входа, а затем пропускает ее через функцию активации ReLU (см. главу 2). Обратите внимание, как `in_layers` используется для передачи предыдущих слоев в качестве входных данных для последующих слоев. Мы закончим построение архитектуры, применяя полностью связанные слои Dense к выходам сверточного слоя. Однако выходные данные слоев Conv2D являются двумерными, поэтому сначала нам нужно применить слой Flatten, чтобы выровнять входные данные в одно измерение. (Точнее, слой Conv2D создает двумерный вывод для каждого образца, поэтому его выход имеет три измерения. Слой Flatten сводит это к одному измерению для образца или к двум измерениям в целом.)

```
flatten = layers.Flatten(in_layers=conv2d_2)
dense1 = layers.Dense(out_channels=1024, activation_fn=tf.nn.relu, in_layers=flatten)
dense2 = layers.Dense(out_channels=10, activation_fn=None, in_layers=dense1)
```

Аргумент `out_channels` в слое Dense указывает ширину слоя. Первый слой выводит 1024 значения на выборку, а второй слой выводит десять значений, соответствующих нашим десяти возможным цифрам. Теперь необходимо подключить этот вывод к функции потерь, чтобы мы могли обучить вывод точно предсказывать классификацию. Для выполнения этой формы обучения мы будем использовать слой потерь `SoftMaxCrossEntropy`:

```
smce = layers.SoftMaxCrossEntropy(in_layers=[label, dense2])
loss = layers.ReduceMean(in_layers=smce)
model.set_loss(loss)
```

Обратите внимание, что слой `SoftMaxCrossEntropy` в качестве входных данных принимает как метки, так и выходные данные последнего слоя `Dense`. Он вычисляет значение функции потерь для каждой выборки, поэтому нам необходимо усреднить все выборки, чтобы получить окончательную потерю. Это делается с помощью слоя `ReduceMean`, который мы устанавливаем как функцию потерь нашей модели, вызывая `model.set_loss()`.

SoftMax и SoftMaxCrossEntropy

Часто бывает необходимо, чтобы модель выводила распределение вероятностей. В случае MNIST мы хотим вывести вероятность того, что данный образец представляет каждую из десяти цифр. Каждый вывод должен быть положительным, и в сумме все выводы должны давать единицу. Простой способ достичь этого – позволить модели вычислить произвольные числа, а затем пропустить их через функцию SoftMax (функцию *мягкого максимума*):

$$\sigma_i(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}.$$

Экспонента в числителе гарантирует, что все значения положительны. Также наличие экспоненты означает, что если один элемент x значительно больше остальных, соответствующий выходной элемент очень близок к 1, а все остальные выходы довольно близки к 0.

`SoftMaxCrossEntropy` сначала использует SoftMax для преобразования выходных данных в вероятности, а затем вычисляет *перекрестную энтропию* этих вероятностей с метками. Помните, что метки имеют унитарное кодирование: 1 для правильного класса, 0 для всех остальных. Мы можем говорить об этом как о распределении вероятностей. Потери сводятся к минимуму, когда прогнозируемая вероятность правильного класса максимально приближена к 1. Эти две операции (SoftMax и перекрестная энтропия) часто появляются вместе, и вычисление их как одного шага оказывается более *численно устойчивым*[1] (numerical stable), чем выполнение их отдельно.

Для обеспечения числовой устойчивости слои, подобные `SoftMaxCrossEntropy`, вычисляются с логарифмическими вероятностями. Нам потребуется преобразовать выходные данные с помощью слоя `SoftMax`, чтобы получить выходные вероятности для каждого класса. Мы добавим этот вывод в модель с помощью метода `model.add_out put()`.

[1] В данном случае подразумевается устойчивость алгоритма к ошибкам округления и случайным флуктуациям, которые мешают правильно классифицировать близкие значения вероятностей. – *Прим. перев.*

```
output = layers.SoftMax(in_layers=dense2)
model.add_output(output)
```

Теперь мы можем приступить к обучению модели, используя ту же функцию `fit()`, которую мы вызывали в предыдущем разделе:

```
model.fit(train_dataset, nb_epoch=10)
```

Обратите внимание, что на стандартном ноутбуке обучение может выполняться достаточно долго! Если вас не устраивает скорость выполнения, попробуйте заменить параметр `nb_epoch=1`. Результаты обучения будут хуже, но вы сможете быстрее завершить оставшуюся часть этой главы.

На этот раз мы определим нашу метрику как *точность*, то есть долю меток, которые правильно спрогнозированы:

```
metric = dc.metrics.Metric(dc.metrics.accuracy_score)
```

Затем мы можем вычислить точность, используя те же вычисления, что и раньше.

```
train_scores = model.evaluate(train_dataset, [metric])
test_scores = model.evaluate(test_dataset, [metric])
```

Мы получили модель отличного качества: точность составляет 0,999 на тренировочном наборе и 0,991 на тестовом наборе. Наша модель правильно идентифицирует более 99 % образцов тестового набора.

Попробуйте воспользоваться GPU

Как мы уже говорили, код глубокого обучения может выполняться довольно медленно! Обучение сверточной нейронной сети на хорошем ноутбуке может занять больше часа. Дело в том, что этот код зависит от большого количества линейных алгебраических операций над данными изображения. Большинство процессоров плохо приспособлено к выполнению таких вычислений.

Если возможно, попробуйте получить доступ к современному графическому процессору (graphics processor unit, GPU). Видеокарты были изначально разработаны для игр, но теперь широко используются для различных вычислений. Большинство современных задач глубокого обучения будет работать на графических процессорах намного быстрее. Примеры, которые вы увидите в этой книге, тоже проще выполнять на GPU.

Если вы не можете использовать GPU, не беспокойтесь. Вы по-прежнему сможете выполнять упражнения, описанные в этой книге. Просто их выполнение займет немного больше времени (возможно, вам придется взять кофе или почитать книгу, пока вы ждете завершения программы).

Заключение

В этой главе вы узнали, как использовать библиотеку DeepChem для создания некоторых простых систем машинного обучения. В оставшейся части данной книги мы продолжим использовать DeepChem в качестве рабочей библиотеки, поэтому не беспокойтесь, если у вас пока нет четкого понимания принципов ее работы. Мы приготовили много других примеров.

В последующих главах мы начнем знакомиться с основными понятиями, необходимыми для эффективного машинного обучения на данных из области наук о жизни. В следующей главе мы познакомим вас с машинным обучением на молекулярных данных.

Глава 4

Машинное обучение и молекулы

В этой главе рассмотрены основы реализации машинного обучения на молекулярных данных. Прежде чем мы углубимся в тему, хотелось бы кратко пояснить, почему машинное обучение в молекулярной области заслуживает отдельного внимания. Большая часть исследований в области современного материаловедения и химии обусловлена необходимостью конструировать новые молекулы, обладающие желаемыми свойствами. Несмотря на то что в поиске новых стратегий проектирования проделана огромная научная работа, иногда для создания интересных молекул все еще требуется много случайного поиска и удача. Мечта о молекулярном машинном обучении состоит в том, чтобы заменить такие случайные эксперименты *управляемым поиском*, где компьютерные *предикторы* (predictor, предсказатель) могут предложить новые молекулы с желаемыми свойствами. Такие предикторы будут быстро создавать принципиально новые материалы и химические вещества с полезными свойствами.

Эта мечта прекрасна, но с чего мы должны начать? Первым шагом является создание технических методов для перевода описаний молекул в векторы чисел, понятные компьютеру. Такие методы называются *молекулярной фичеризацией* (molecular featurization), или *выделением молекулярных признаков*. Молекулы являются сложными объектами, и исследователи разработали множество различных методов их компьютерного представления. Эти представления включают в себя векторы химических дескрипторов, представления в виде *двухмерного графа* (2D graph), в виде *пространственного электростатического распределения* (3D electrostatic grid), на основе *орбитальной функции* (orbital function) и другие. Мы начнем рассмотрение методов фичеризации в текущей главе и еще больше методов изучим в главе 5.

После фичеризации молекула становится обучающим материалом. Мы рассмотрим некоторые алгоритмы обучения на молекулах, включая простые полностью связанные сети, и более сложные методы, такие как *графовая свертка*. Мы также опишем некоторые ограничения технологии графовой свертки и поясним, что мы можем и чего не можем ожидать от нее. Завершает главу учебный пример молекулярного машинного обучения на интересном наборе данных.

Что такое молекула?

Прежде чем мы углубимся в молекулярное машинное обучение, будет полезно разобраться, что именно представляет собой *молекула*. Этот вопрос звучит немного

глупо, поскольку такие молекулы, как H_2O и CO_2, знакомы даже маленьким детям. Разве ответ не очевиден? Однако факт в том, что подавляющее большинство людей понятия не имеют, как *на самом деле* устроены молекулы. Рассмотрим мысленный эксперимент: как бы вы убедили скептически настроенного иностранца в том, что невидимые сущности, называемые молекулами, существуют? Это непростая задача. Например, вам может понадобиться масс-спектрометр.

Масс-спектроскопия

Иногда бывает довольно сложно идентифицировать молекулы, входящие в состав заданного образца. В настоящее время наиболее популярным методом является *масс-спектроскопия*, основная идея которой заключается в бомбардировке образца электронами. Поток ускоренных электронов разбивает молекулы на фрагменты. Эти фрагменты обычно ионизируются, то есть захватывают или теряют электроны, и становятся заряженными. Эти заряженные фрагменты приводятся в движение электрическим полем, а траектория движения зависит от отношения массы к заряду. Пространственное распределение заряженных фрагментов называется *спектром*. Процесс схематически показан на рис. 4.1. По набору обнаруженных фрагментов часто удается идентифицировать молекулы, которые были в исходном образце. Однако этот процесс все еще сложен и недостаточно точен. Ряд исследователей активно работает над улучшением масс-спектроскопии с помощью алгоритмов глубокого обучения, чтобы упростить идентификацию исходных молекул по спектру заряженных фрагментов.

Не забывайте о сложности этого исследования! Молекулы – это особенные объекты, которые трудно идентифицировать.

Для начала давайте дадим определение молекулы как группы атомов, объединенных физическими силами. Молекула – это самая маленькая фундаментальная единица химического соединения, которая может участвовать в химической реакции. Атомы в молекуле связаны друг с другом *химическими связями*, которые удерживают атомы вместе и ограничивают их движение относительно друг друга. Размеры молекул варьируются в широчайшем диапазоне, от нескольких штук атомов до многих тысяч атомов. Простое схематическое изображение молекулы показано на рис. 4.2.

Опираясь на это базовое представление о молекулах, в следующих двух разделах мы углубимся в различные понятия молекулярной химии. Не беда, если вы не усвоите все идеи при первом чтении этой главы, но всегда полезно иметь под рукой базовые знания в области химии.

Молекулы – это динамические квантовые объекты

В приведенном выше определении мы даем упрощенное описание молекул в терминах атомов и связей. Очень важно помнить, что внутри каждой молекулы происходит много разных событий. Молекулы являются динамическими объектами, поэтому все атомы в данной молекуле находятся в быстром движении относительно друг друга. Сами связи способны растягиваться вперед и назад и, возможно, быстро колеблются в длину. Обычно атомы отрываются от молекул и тут же присоединяются обратно. Мы расскажем немного больше о динамической природе молекул при обсуждении молекулярных конформаций.

Еще более странно, что молекулы являются квантовыми объектами. Квантовую сущность объекта можно пояснить многими способами, но если говорить проще, то «атомы» и «связи» в молекуле выражены гораздо менее четко, чем кажется при взгляде на схему из шариков и палочек. Атомы и связи – это достаточно общие определения. На этом этапе от вас не требуется глубокое понимание, просто помните, что наши изображения молекул очень условны. Это имеет практическое значение, поскольку для разных учебных задач может потребоваться разное представление молекул.

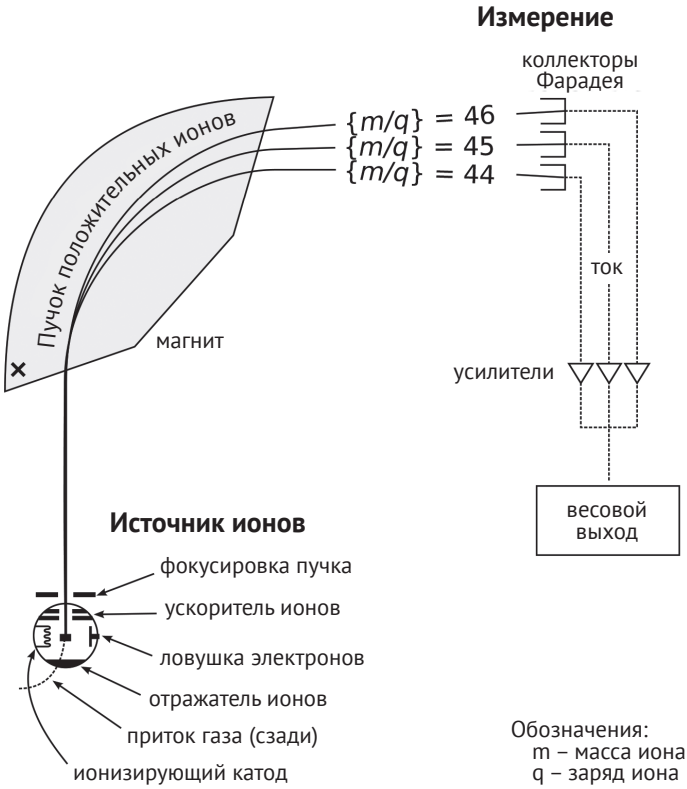


Рис. 4.1 ❖ Упрощенная схема масс-спектрометра.
Источник: **https://commons.wikimedia.org/wiki/File:Mass_Spectrometer_Schematic.svg**

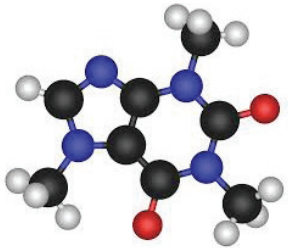


Рис. 4.2 ❖ Простое представление молекулы кофеина в виде структуры из шариков и стержней. Атомы представлены в виде цветных шариков (черный – углерод, красный – кислород, синий – азот, белый – водород), соединенных стержнями, которые изображают химические связи

Что такое внутримолекулярные связи?

Если вы не химик, то наверняка изучали основы химии только в школе, но с тех пор прошло много времени. Поэтому сейчас мы вернемся к изучению основных химических понятий. Самый главный вопрос: что такое *химическая связь*?

Молекулы, составляющие окружающий мир, состоят из атомов, часто очень большого их количества. Эти атомы соединены между собой химическими связями, которые, по существу, «склеивают» атомы своими общими электронами. Существует много различных типов внутримолекулярных связей, включая ковалентные связи и несколько типов нековалентных связей.

Ковалентные связи

Ковалентные связи (covalent bonds) представляют собой обмен электронами между двумя атомами, так что одни и те же электроны принадлежат обоим атомам. В целом ковалентные связи являются наиболее сильным типом химической связи. Они образуются и разрушаются в химических реакциях. После образования ковалентных связей требуется много энергии, чтобы разорвать их, поэтому атомы могут оставаться связанными в течение очень долгого времени. Вот почему молекулы ведут себя как отдельные объекты, а не как сгустки несвязанных атомов. Фактически ковалентные связи определяют молекулу, то есть молекула представляет собой набор атомов, соединенных ковалентными связями.

Рис. 4.3 ❖ Слева: два атомных ядра, каждое из которых окружено облаком электронов. Справа: когда атомы сближаются, электроны начинают проводить больше времени в пространстве между ядрами, образуя ковалентную связь между атомами

Нековалентные связи

Нековалентные связи (noncovalent bonds) не предполагают прямого обмена электронами между атомами и представляют собой более слабые электромагнитные взаимодействия. Поскольку они не так сильны, как ковалентные связи, они более эфемерны, постоянно разрываются и перестраиваются. Нековалентные связи не определяют молекулы в том же смысле, в каком это делают ковалентные связи, но они оказывают огромное влияние на определение формы, которую принимают молекулы, и способов, которыми разные молекулы связываются друг с другом.

«Нековалентные связи» – это общий термин, охватывающий несколько различных типов взаимодействий, например *водородные связи* (hydrogen bonds), *соляные мостики* (salt bridges), *π-стекинг* (pi-stacking) и некоторые другие. Эти типы взаимодействий часто играют решающую роль в разработке лекарств, которые взаимодействуют с биологическими молекулами в организме человека в основном посредством нековалентных связей.

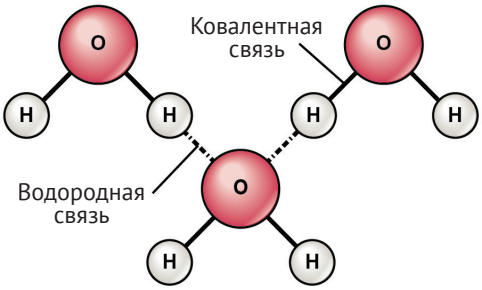


Рис. 4.4 ❖ Молекулы воды имеют сильные водородные связи между атомами водорода и кислорода соседних молекул. Сеть сильных водородных связей частично способствует превращению воды в растворитель. *Источник*: **https://commons.wikimedia.org/wiki/File:SimpleBayesNet.svg**

В различных главах книги мы столкнемся с каждым из упомянутых типов связей. В этой главе мы в основном будем иметь дело с ковалентными связями, но когда мы начнем изучать некоторые глубокие биофизические модели, нековалентные взаимодействия будут для нас намного важнее.

Молекулярные графы

Граф – это математическая структура данных, состоящая из *узлов* (nodes), соединенных *ребрами* (edges). Графы являются невероятно полезным абстрактным понятием в информатике. На самом деле существует целый раздел математики под названием «теория графов», посвященный изучению свойств графов и нахождению способов работы с ними. Графы используются для описания всего: от компьютеров, составляющих сеть, до пикселей, из которых состоит изображение, и актеров, снимавшихся в фильмах с Кевином Бэконом.

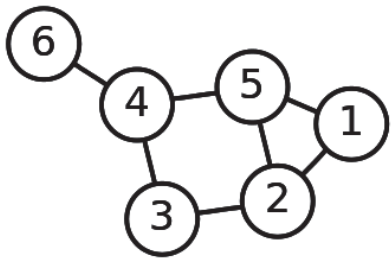


Рис. 4.5 ❖ Пример математического графа с шестью узлами, соединенными ребрами. *Источник*: **https://commons.wikimedia.org/wiki/File:6n-graf.svg**

Важно отметить, что молекулы также можно рассматривать как графы. В этом представлении атомы являются узлами, а химические связи являются ребрами графа. Любая молекула может быть преобразована в соответствующий *молекулярный граф* (molecular graph).

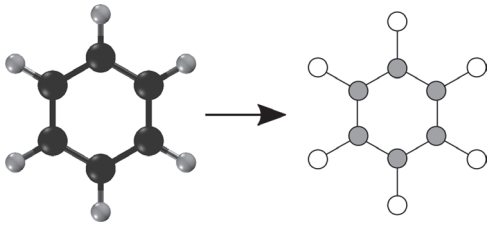


Рис. 4.6 ❖ Пример преобразования молекулы бензола в молекулярный граф. Обратите внимание, что атомы превращаются в узлы, а химические связи – в ребра

Далее в этой главе мы будем неоднократно преобразовывать молекулы в графы, чтобы анализировать их и учиться делать предсказания.

Конформации молекулы

Молекулярный граф описывает набор атомов в молекуле и то, как они связаны друг с другом. Но есть еще одна очень важная вещь, которую мы также должны знать, – как атомы расположены друг относительно друга в трехмерном пространстве. Это расположение называется *конформацией* молекулы (conformation).

Разумеется, понятия графа и конформации связаны друг с другом. Ковалентная связь двух атомов фиксирует расстояние между ними, сильно ограничивая возможные конформации. Углы, образованные группами из трех или четырех связанных атомов, также часто ограничены. Иногда встречаются целые скопления атомов, которые жестко связаны и движутся вместе как единое целое. Но бывают и гибкие части молекул, что позволяет атомам двигаться относительно друг друга. Например, многие (но не все) ковалентные связи позволяют группам атомов, которые они соединяют, свободно вращаться вокруг оси связи. Это дает возможность молекуле принимать множество различных конформаций.

На рис. 4.7 показана очень популярная молекула, сахароза, также известная как столовый сахар. Она показана как в виде трехмерной конформации, так и в виде двухмерной химической структуры. Сахароза состоит из двух колец, соединенных вместе. Каждое из колец довольно жесткое, поэтому их форма почти не меняется. Но соединяющий их *линкер* гораздо гибче, он позволяет кольцам двигаться относительно друг друга.

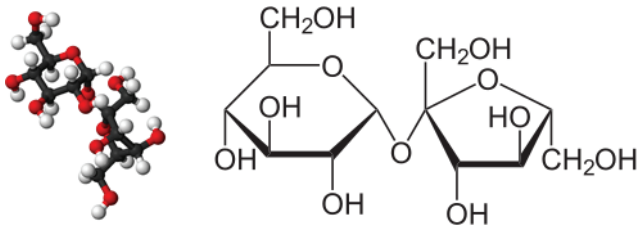


Рис. 4.7 ❖ Трехмерная конформация молекулы сахарозы (слева) и двухмерная структура сахарозы (справа). *Источники*: **https://commons.wikimedia.org/wiki/File:Sucrose-3D-balls.png**, **https://en.wikipedia.org/wiki/File:Saccharose2.svg**

По мере того как молекулы становятся больше, количество возможных конформаций, которые они могут принять, стремительно возрастает (рис. 4.8). Для больших макромолекул, таких как белки, моделирование множества возможных конформаций в настоящее время требует очень дорогих вычислительных ресурсов.

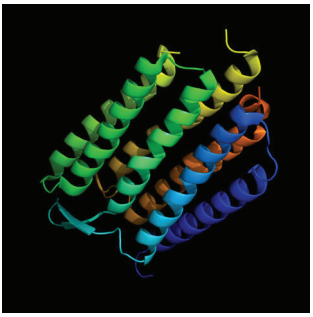


Рис. 4.8 ❖ На этом рисунке представлена 3D-конформация бактериородопсина (белок, преобразующий световую энергию в химическую). Конформации белка обладают множеством трехмерных геометрических фрагментов и служат хорошим напоминанием о том, что у молекул, кроме химических формул, есть еще и геометрическая структура. *Источник*: **https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/1M0K.png/480px-1M0K.png**

Хиральность молекул

Некоторые молекулы (в том числе многие лекарства) бывают двух видов, которые являются зеркальным отображением друг друга. Это явление называется *хиральностью* (chirality). Хиральная молекула имеет как «правую» форму (также известную как R-форма), так и «левую» форму (также известную как S-форма). На рис. 4.9 показан пример аксиальной хиральности *спиросоединения* (соединения, состоящего из двух или более колец, соединенных вместе). Обратите внимание, что два хиральных варианта обозначены как R и S соответственно. Это соглашение широко распространено в химической литературе.

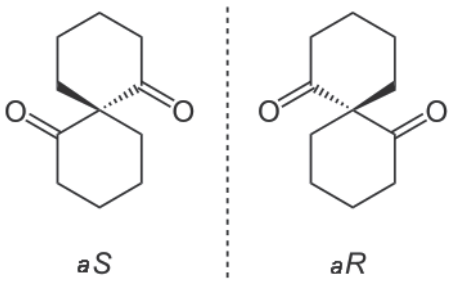


Рис. 4.9 ❖ Пример аксиальной хиральности спиросоединения (соединение, образованное двумя или более кольцами, связанными воедино). Обратите внимание, что хиральные варианты помечены символами R и S. Это соглашение широко применяется в химической литературе

Хиральность очень важна, а также очень огорчает как лабораторных химиков, так и специалистов по *хемоинформатике* (chemoinformatics). Начнем с того, что химические реакции, которые производят хиральные молекулы, часто не различают формы, производя обе хиральности в равных количествах. Такие продукты называются *рацемическими смесями* (racemic mixtures). Поэтому, если вы хотите получить только одну форму, ваш производственный процесс резко усложняется. Кроме того, многие физические свойства продукта одинаковы для обеих хиральностей, поэтому хиральные версии молекулы зачастую экспериментально неразличимы. То же самое относится и к вычислительным моделям. Например, у обеих хиральных форм одинаковые молекулярные графы, поэтому любая модель машинного обучения, зависящая только от молекулярного графа, не сможет различить их.

Это не имело бы большого значения, если бы хиральные формы вели себя на практике одинаково, но часто это не так. Две хиральные формы лекарства будут связываться с разными белками организма и давать совершенно разные эффекты. Во многих случаях только одна форма лекарственного средства обладает желаемым терапевтическим эффектом. Другая форма просто вызывает неприятные побочные эффекты без какой-либо выгоды для больного.

Одним из конкретных примеров различного действия хиральных соединений является препарат талидомид, который назначался в качестве седативного средства в 1950-х и 1960-х годах. Впоследствии этот препарат продавался без рецепта как средство от тошноты и утреннего недомогания, связанного с беременностью. R-форма талидомида является эффективным седативным средством, тогда как S-форма является *тератогенной*[1] (teratogenic) и, как было показано, вызывает серьезные врожденные дефекты. Эти трудности еще более осложняются тем фактом, что талидомид взаимопревращается, или рацемизируется между двумя различными формами непосредственно в организме.

Фичеризация молекулы

Как мы можем провести фичеризацию молекул на основании представлений об их внутреннем устройстве? То есть как представить молекулы таким образом, чтобы их можно было использовать в качестве входных данных для обучения модели? В этой главе мы рассмотрим ряд различных способов компьютерного представления молекул.

Чтобы выполнить машинное обучение на молекулах, нам нужно преобразовать их в векторы признаков, которые можно использовать в качестве входных данных для моделей. В этом разделе мы обсудим подмодуль фичеризации DeepChem `dc.feat` и объясним, как его использовать для фичеризации молекул разными способами.

Строки SMILES и пакет RDKit

SMILES – это популярный метод описания молекул с помощью текстовых строк. Название является аббревиатурой строки «Simplified Molecular-Input Line-Entry

[1] Вызывает патологические нарушения во время развития зародыша или плода. – *Прим. перев.*

System» (система упрощенного представления молекул в строке ввода). Кто-то явно очень постарался, придумывая это странное название. Тем не менее строка SMILES описывает атомы и связи молекулы одновременно точно и достаточно интуитивно понятно для химиков. Для людей, далеких от химии, строки SMILES выглядят как бессмысленные наборы случайных символов. Например, строка «*OCCc1c*(*C*)*[n+]*(*cs1*)*Cc2cnc*(*C*)*nc2N*» описывает важный питательный элемент тиамин, также известный как витамин B1.

DeepChem использует строки SMILES в качестве формата для представления молекул внутри наборов данных. Существуют некоторые модели глубокого обучения, которые напрямую принимают строки SMILES в качестве входных данных, пытаясь научиться находить значимые признаки в текстовом представлении. Но гораздо чаще мы сначала конвертируем строку в другое представление, более подходящее для рассматриваемой проблемы. Этот процесс – преобразование выборок из представления, хранящегося в наборе данных, в представление, требуемое моделью, – мы называем *фичеризацией*.

DeepChem зависит от другого пакета хемоинформатики с открытым исходным кодом, RDKit, предназначенного для облегчения работы с молекулами. Пакет RDKit предоставляет множество функций для работы со строками SMILES. Он играет центральную роль в преобразовании строк набора данных в молекулярные графы и другие представления, описанные ниже.

Расширенные отпечатки связей

Алгоритм *дополненных отпечатков связей* (Extended Connectivity Fingerprints, ECFP) представляет собой класс функций, которые сочетают в себе несколько полезных свойств. Они берут молекулы произвольного размера и превращают их в векторы данных фиксированной длины. Это очень важно, поскольку многие модели требуют, чтобы их входные данные имели одинаковый размер. ECFP позволяют брать молекулы разных размеров и использовать их с одной и той же моделью. ECFP также очень легко сравнивать. Вы можете просто взять отпечатки для двух молекул и сравнить соответствующие элементы. Чем больше совпадающих элементов, тем более похожи молекулы. Наконец, ECFP быстро вычисляются.

Каждый элемент вектора отпечатка указывает на наличие или отсутствие определенного молекулярного признака, определяемого некоторым локальным расположением атомов. Алгоритм начинается с рассмотрения каждого атома по отдельности и рассмотрения нескольких свойств атома: номера химического элемента, числа ковалентных связей, которые он образует, и т. д. Каждая уникальная комбинация этих свойств представляет собой признак. Наличие определенного признака обозначается единицей в векторе. Затем алгоритм выходит на уровень выше, описывая связи каждого атома с другими атомами. Таким образом, определяется новый набор более крупных признаков, и устанавливаются в единицу соответствующие элементы вектора. Наиболее распространенным вариантом этого метода является алгоритм ECFP4, который охватывает радиус двух связей вокруг центрального атома.

Библиотека RDKit предоставляет утилиты для вычисления отпечатков связей по алгоритму ECFP4. В свою очередь, библиотека DeepChem предоставляет удобные обертки для этих функций. Класс `dc.feat.CircularFingerprint` наследуется от `Featurizer` и предоставляет стандартный интерфейс фичеризации молекул:

```
smiles = ['C1CCCCC1', 'O1CCOCC1'] # Циклогексан и диоксан
mols = [Chem.MolFromSmiles(smile) for smile in smiles]
feat = dc.feat.CircularFingerprint(size=1024)
arr = feat.featurize(mols)
# arr – это массив 2×1024, содержащий отпечатки
# двух молекул
```

У ECFP есть один важный недостаток, связанный с потерей информации. Две разные молекулы могут иметь одинаковые отпечатки. Имея только отпечаток, невозможно однозначно определить, из какой молекулы он получен. Отпечаток содержит большое количество информации о молекуле, но некоторая информация теряется.

Молекулярные дескрипторы

Альтернативный подход гласит, что молекулы удобно описывать с помощью набора физико-химических дескрипторов. Они обычно соответствуют различным вычисленным величинам, которые описывают структуру молекулы. Эти величины, такие как *логарифм коэффициента распределения* (log partition coefficient) или *полярная площадь поверхности* (polar surface area), часто выводятся из определений классической физики или химии. Пакет RDKit вычисляет много подобных физических дескрипторов молекул. Фичеризатор DeepChem `dc.feat.RDKitDescriptors()` предоставляет простой способ реализации тех же вычислений:

```
feat = dc.feat.RDKitDescriptors()
arr = feat.featurize(mols)
# arr – это массив 2×111, содержащий свойства
# двух молекул
```

Эта функция в разной степени полезна для разных проблем. Она лучше всего работает для предсказания вещей, зависящих от относительно общих свойств молекул. Вряд ли можно применить эту функцию для прогнозирования свойств, которые зависят от детального расположения атомов.

Графовые свертки

Описанные выше фичеризации были разработаны людьми. Эксперт тщательно продумал, как представить молекулы таким образом, чтобы их можно было использовать в качестве входных данных для моделей машинного обучения, а затем закодировал представление вручную. Можем ли мы позволить модели выяснить для себя лучший способ представления молекул? Это ведь машинное обучение, в конце концов! Вместо того чтобы самим разрабатывать фичеризацию, можно попробовать научить модель делать ее автоматически на основе данных.

В качестве аналогии рассмотрим сверточную нейронную сеть для распознавания изображений. Входом в сеть является необработанное изображение. Оно состоит из вектора чисел для каждого пикселя, например трех цветовых компонентов. Это очень простое, предельно общее представление изображения. Первый сверточный слой учится распознавать простые шаблоны, такие как вертикальные или горизонтальные линии. Его вывод снова представляет собой вектор чисел для

каждого пикселя, но теперь он представлен более абстрактно. Каждое число представляет наличие некоторой локальной геометрической особенности.

Сеть представляет собой последовательность слоев. Каждый слой выводит новое представление изображения, более абстрактное, чем представление предыдущего уровня, и менее тесно связанное с необработанными цветными компонентами. И эти представления автоматически извлекаются из данных, а не разрабатываются человеком. Никто не говорит модели, какие образцы искать, чтобы определить, содержит ли изображение кошку. Модель сама это выясняет благодаря обучению.

Графовая сверточная сеть (graph convolutional network) берет эту же идею и применяет ее к графам. Так же как обычная сверточная сеть начинается с вектора чисел для каждого пикселя, графовая сверточная сеть начинается с вектора чисел для каждого узла и/или ребра. Когда граф представляет молекулу, эти числа могут быть химическими свойствами высокого уровня каждого атома, такими как его номер элемента, заряд и *степень гибридизации орбиталей* (hybridization state). Так же как обычный сверточный слой вычисляет новый вектор для каждого пикселя на основе локальной области своего ввода, графовый сверточный слой вычисляет новый вектор для каждого узла и/или ребра. Выходные данные вычисляются путем применения сверточного ядра к каждой локальной области графа, где понятие «локальный» теперь определяется в терминах ребер между узлами. Например, он может вычислить выходной вектор для каждого атома на основе входного вектора для того же атома и любых других атомов, с которыми он непосредственно связан.

Это общая идея. Когда дело дошло до деталей, было предложено много разных вариантов. К счастью, DeepChem включает в себя реализации многих графовых архитектур, так что вы можете попробовать их, даже не разбираясь во всех деталях.

В качестве примеров можно привести графовые свертки (GraphConvModel), WAVE-модели (WeaveModel), нейронные сети с обменом сообщениями (MPNNModel), тензорированные глубокие нейросети (DTNNModel) и др.

Графовые сверточные сети являются мощным инструментом для анализа молекул, но у них есть одно важное ограничение: расчет основан исключительно на молекулярном графе. Они не получают данных о конформации молекулы, поэтому не могут предсказать что-либо, зависящее от конформации. Это ограничение делает графовые сверточные сети наиболее подходящими для маленьких, в основном жестких молекул. В следующей главе мы обсудим методы, которые больше подходят для крупных гибких молекул, которые могут принимать множество конформаций.

Обучение модели для прогнозирования растворимости

Давайте соединим все, что узнали, воедино и обучим модель на реальном химическом наборе данных, чтобы предсказать важное молекулярное свойство. Сначала загрузим данные:

```
tasks, datasets, transformers = dc.molnet.load_delaney(featurizer='GraphConv')
train_dataset, valid_dataset, test_dataset = datasets
```

Этот набор данных содержит информацию о *растворимости* (solubility), которая является мерой того, насколько легко вещество растворяется в воде. Это свойство жизненно важно для любого химического вещества, которое вы надеетесь использовать в качестве лекарства. Если оно растворяется с трудом, то может не попасть в кровоток пациента для оказания терапевтического эффекта. Химики-фармацевты проводят много времени, модифицируя молекулы для увеличения растворимости препаратов.

Обратите внимание, что мы указываем опцию `featurizer='GraphConv'`. Мы собираемся использовать графовую сверточную модель, и эта опция говорит MoleculeNet преобразовать SMILES-строку каждой молекулы во входной формат модели. Теперь давайте построим и обучим модель.

```
model = GraphConvModel(n_tasks=1, mode='regression', dropout=0.2)
model.fit(train_dataset, nb_epoch=100)
```

Мы указываем, что существует только одна задача, то есть одно выходное значение (растворимость) для каждого образца. Мы также указываем, что это регрессионная модель, подразумевающая, что метки являются непрерывными числами и модель должна стараться воспроизвести их как можно точнее. Этим она отличается от классифицирующей модели, которая пытается предсказать принадлежность каждой выборки к определенному элементу из набора классов. Чтобы уменьшить переобучение, мы указываем коэффициент исключения 0,2, означающий, что 20 % выходных сигналов от каждого сверточного слоя будут случайным образом обнулены.

Вот и все, что нужно сделать! Теперь мы можем оценить модель и посмотреть, насколько хорошо она работает. В качестве метрики оценки мы будем использовать *коэффициент корреляции Пирсона* (Pearson correlation coefficient):

```
metric = dc.metrics.Metric(dc.metrics.pearson_r2_score)
print(model.evaluate(train_dataset, [metric], transformers))
print(model.evaluate(test_dataset, [metric], transformers))
```

В нашем случае коэффициент корреляции составляет 0,95 для тренировочного набора и 0,83 для проверочного набора. Видимо, наблюдается небольшое переобучение, но это не страшно. И коэффициент корреляции 0,83 вполне хорош. Наша модель успешно предсказывает растворимость молекул на основе их молекулярных структур!

Теперь, когда у нас есть модель, мы можем использовать ее для прогнозирования растворимости новых молекул. Предположим, мы заинтересованы в следующих пяти молекулах, заданных строками SMILES:

```
smiles = ['COC(C)(C)CCCC(C)CC=CC(C)=CC(=O)OC(C)C',
          'CCOC(=O)CC',
          'CSc1nc(NC(C)C)nc(NC(C)C)n1',
          'CC(C#C)N(C)C(=O)Nc1ccc(Cl)cc1',
          'Cc1cc2ccccc2cc1C']
```

Чтобы использовать их в качестве входных данных для модели, мы должны сначала использовать RDKit для анализа строк SMILES, а затем использовать средство настройки DeepChem для преобразования их в формат, ожидаемый графовой сверткой.

```
from rdkit import Chem
mols = [Chem.MolFromSmiles(s) for s in smiles]
featurizer = dc.feat.ConvMolFeaturizer()
x = featurizer.featurize(mols)
```

Теперь мы можем передать молекулы модели и попросить ее предсказать их растворимость:

```
predicted_solubility = model.predict_on_batch(x)
```

MoleculeNet

Сейчас мы знакомы с двумя наборами данных, загруженных из модуля `molnet`: набор данных о токсичности Tox21 в предыдущей главе и набор данных о растворимости Delaney в этой главе.

MoleculeNet – это большая коллекция наборов данных, полезных для молекулярного машинного обучения. Как показано на рис. 4.10, коллекция содержит данные о многих видах молекулярных свойств. Они варьируются от физических свойств низкого уровня, которые можно рассчитать с помощью квантовой механики, до информации очень высокого уровня об их взаимодействиях с организмом человека, таких как токсичность и побочные эффекты.

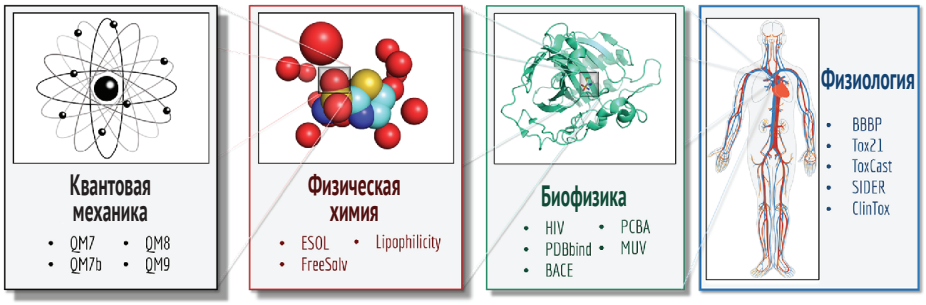


Рис. 4.10 ❖ MoleculeNet содержит множество различных наборов данных из разных молекулярных наук. Ученые считают полезным предсказывать квантовые, физико-химические, биофизические и физиологические качества молекул

При разработке новых методов машинного обучения вы можете использовать MoleculeNet в качестве набора стандартных тестов для проверки своего метода. На сайте **http://moleculenet.ai** вы можете просмотреть данные о том, насколько хорошо работает набор стандартных моделей для каждого из наборов данных, и получить представление о том, насколько ваш собственный метод сопоставим с общепринятыми методами.

Строки SMARTS

Когда мы занимаемся обработкой текста, нам часто приходится искать определенную текстовую строку. В хемоинформатике мы сталкиваемся с похожими ситуациями, когда нужно определить, расположены ли атомы в молекуле по определенной схеме, например:

- поиск в базе данных молекул, содержащих определенную подструктуру;
- сортировка набора молекул по общей подструктуре для улучшения визуализации;
- подсветка подструктуры на схематическом изображении молекулы;
- ведение расчетов только в рамках подструктуры.

SMARTS – это расширение языка SMILES, описанного выше, применяемое для создания запросов. Строки SMARTS похожи на регулярные выражения для поиска по тексту. Например, при поиске по имени файла можно указать запрос типа «foo*.bar», который будет соответствовать именам foo.bar, foo3.bar и foolish.bar. На низком уровне любая строка SMILES также может быть строкой SMARTS. Строка SMILES «CCC» одновременно является строкой SMARTS и будет соответствовать последовательностям трех соседних атомов углерода в алифатическом соединении. Давайте рассмотрим пример кода, показывающий, как мы можем определять молекулы из строк SMILES, отображать эти молекулы и выделять атомы, соответствующие шаблону SMARTS.

Сначала мы импортируем необходимые библиотеки и создадим список молекул в виде списка строк SMILES. Результат показан на рис. 4.11.

```
from rdkit import Chem
from rdkit.Chem.Draw import MolsToGridImage
smiles_list = ["CCCCC","CCOCC","CCNCC","CCSCC"]
mol_list = [Chem.MolFromSmiles(x) for x in smiles_list]
```

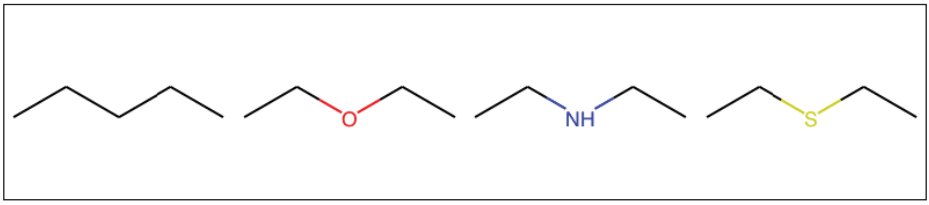


Рис. 4.11 ❖ Химическая структура, сгенерированная из списка SMILES

Теперь мы можем узнать, какие строки SMILES соответствуют шаблону SMARTS «CCC» (рис. 4.12):

```
query = Chem.MolFromSmarts("CCC")
match_list = [mol.GetSubstructMatch(query) for mol in
              mol_list]
MolsToGridImage(mols=mol_list, molsPerRow=4,
                highlightAtomLists=match_list)
```

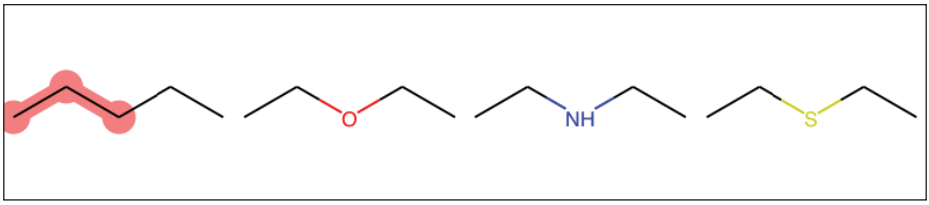


Рис. 4.12 ❖ Молекулы, удовлетворяющие требованию «CCC»

На рис. 4.12 нужно отметить несколько нюансов. Во-первых, выражение SMARTS соответствует только первой структуре. Другие структуры не содержат три атома углерода подряд. Следует также отметить, что шаблон SMARTS мог бы соответствовать первой молекуле на этом рисунке несколькими разными способами. Он может соответствовать любым трем соседним атомам углерода, начиная с первого, второго или третьего атома углерода в молекуле. В RDKit есть дополнительные функции, которые будут возвращать все возможные совпадения SMARTS, но мы не будем их обсуждать.

В шаблоне последовательности атомов могут использоваться дополнительные символы подстановки. Как и в случае с текстом, символ «*» означает соответствие любому атому. Например, строка SMARTS «C*C» будет соответствовать любому атому, расположенному между двумя атомами углерода (рис. 4.13).

```
query = Chem.MolFromSmarts("C*C")
match_list = [mol.GetSubstructMatch(query) for mol in
              mol_list]
MolsToGridImage(mols=mol_list, molsPerRow=4,
                highlightAtomLists=match_list)
```

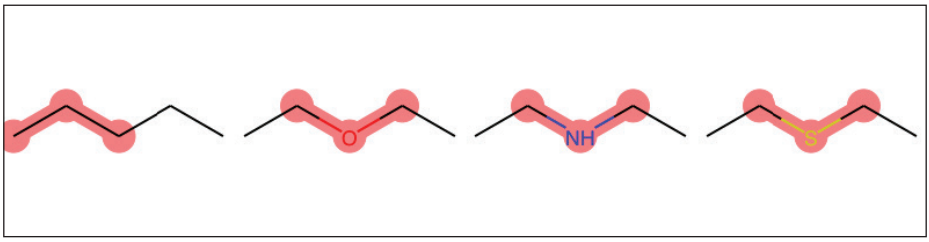


Рис. 4.13 ❖ Молекулы, соответствующие SMARTS-выражению «C*C»

Синтаксис SMARTS можно расширить, чтобы разрешить только определенные наборы атомов. Например, строка «C[C,O,N]C» будет соответствовать атому углерода, кислорода или азота, расположенному между двумя атомами углерода (рис. 4.14).

```
query = Chem.MolFromSmarts("C[C,N,O]C")
match_list = [mol.GetSubstructMatch(query) for mol in
              mol_list]
MolsToGridImage(mols=mol_list, molsPerRow=4,
                highlightAtomLists=match_list)
```

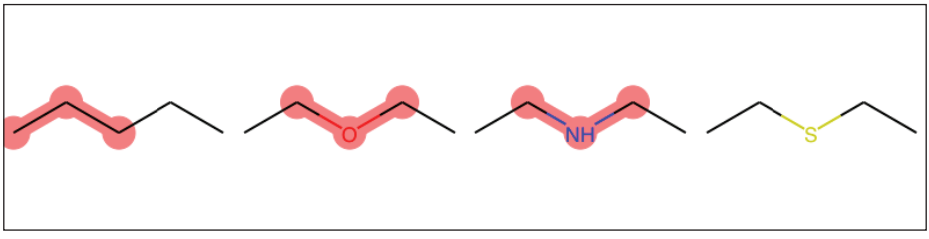


Рис. 4.14 ❖ Молекулы, соответствующие SMARTS-выражению «C[C,N,O]C»

Полное описание SMARTS выходит за рамки этого краткого введения. Чтобы глубже понять SMILES и SMARTS, заинтересованным читателям настоятельно рекомендуется прочитать руководство[1], изданное компанией Daylight Chemical Information Systems. Как мы увидим в главе 11, SMARTS можно использовать для создания сложных запросов, помогающих идентифицировать трудноуловимые молекулы в биологических анализах.

Заключение

В этой главе вы познакомились с основами молекулярного машинного обучения. В частности, освежили в памяти основы химии и узнали о способах представления молекул в вычислительных системах. Вы также узнали о графовых свертках, которые являются новым подходом к моделированию молекул в глубоком обучении. На этих знаниях основан полный рабочий пример использования молекулярного машинного обучения для прогнозирования важного физического свойства – растворимости вещества. В последующих главах мы будем неоднократно возвращаться к методам, упомянутым в этой главе.

[1] **https://www.daylight.com/dayhtml/doc/theory/**.

Глава 5

Глубокое обучение и биофизика

В этой главе мы рассмотрим, как глубокое обучение применяется для исследования биофизических систем. В частности, мы подробно изучим проблему прогнозирования взаимодействий молекул лекарственного вещества с важными белками человеческого организма.

Эта проблема имеет принципиальное значение при создании новых лекарств. Целенаправленное воздействие на конкретный белок зачастую дает значительный терапевтический эффект. Так, например, прорывной препарат против рака Imatinib прочно связывается с белком BCR-ABL, что является одной из причин его эффективности. Для других заболеваний может быть сложно найти единственный белок-мишень с такой же эффективностью, но тем не менее подобный подход весьма перспективен. В человеческом теле действует так много механизмов, что поиск эффективной ментальной модели может иметь решающее значение.

Лекарства взаимодействуют с разными белками

Как мы уже говорили выше, иногда бывает чрезвычайно полезно свести проблему разработки лекарственного средства к поиску вещества, которое тесно взаимодействует с нужным белком. Но следует понимать, что в действительности любое лекарство взаимодействует со многими подсистемами организма. Изучение комплекса таких многогранных взаимодействий называется *полифармакологией*.

В настоящее время вычислительные методы в полифармакологии все еще недостаточно разработаны, поэтому золотым стандартом для тестирования полифармакологических эффектов остаются эксперименты на животных и людях. По мере развития вычислительных методов это положение дел может измениться в течение следующих нескольких лет.

Поэтому наша цель – разработать алгоритмы машинного обучения, которые могут достаточно точно предсказать взаимодействие молекулы с определенным белком. Как мы можем это сделать? Для начала мы могли бы позаимствовать некоторые методы из предыдущей главы о молекулярном машинном обучении и попытаться создать специфическую модель белка. На основе обучающего набора данных о *связываемости* (binding) известных молекул с данным белком такая модель научилась бы предсказывать связываемость для новых молекул. Эта идея на самом деле не столь сложна, как может показаться, но требует наличия большого количества обучающих данных. В идеале, мы постепенно получили бы алгоритм, способный работать с новыми белками при ограниченном объеме обучающих данных.

Как оказалось, проблема заключается не столько в химии, сколько в физике белка. Как мы покажем в следующем разделе, о физической структуре белковых молекул известно совсем немного. В частности, современные экспериментальные методы позволяют делать снимки трехмерной структуры белков. Эти *объемные снимки* (3D snapshots) могут быть введены в алгоритмы обучения и использованы для прогнозирования связываемости. Также можно получить снимки взаимодействия белков с более мелкими молекулами, так называемыми *лигандами* (ligands). Если эти рассуждения кажутся вам оторванными от информатики, не беспокойтесь. В данной главе вы увидите достаточно много прикладного кода.

Мы начнем эту главу с углубленного обзора белков и их функций в биологии. Затем вернемся к информатике и представим некоторые *алгоритмы фичеризации белковых систем*, которые могут преобразовывать биофизические системы в векторы или тензоры для использования в машинном обучении. Глава завершается детальным изучением практического примера разработки модели взаимодействия *белок–лиганд*. Для экспериментов мы воспользуемся набором данных PDBBind, который содержит коллекцию экспериментально определенных структур белок–лиганд. Мы продемонстрируем, как провести фичеризацию этого набора данных с помощью DeepChem. Затем на основе фичеризованных наборов данных мы построим различные модели, как глубокие, так и более простые, и изучим их точность.

Почему это называется биофизикой?

Часто говорят, что вся биология основана на химии, а вся химия основана на физике. На первый взгляд, биология и физика достаточно далеки друг от друга. Но позже в этой главе мы покажем, что в основе всех биологических механизмов лежат физические законы. Кроме того, большая часть важнейших экспериментальных методов изучения структуры белка разработана физиками. Чтобы манипулировать наноразмерными системами (чем на самом деле являются белки), необходимо в совершенстве знать теоретическую и прикладную физику.

Также интересно отметить, что алгоритмы глубокого обучения, которые мы обсудим в этой главе, схожи с архитектурами глубокого обучения в области физики элементарных частиц или физического моделирования. Такие темы выходят за рамки этой книги, но мы рекомендуем заинтересованным читателям продолжить их изучение.

Белковые структуры

Белки – это крошечные машины, которые выполняют большую часть работы в клетке. Несмотря на их небольшой размер, они могут быть очень сложными. Типичный белок состоит из тысяч атомов, расположенных строго определенным образом.

Чтобы понять суть любой машины, вы должны знать, из каких частей она состоит и как они взаимодействуют. Вы не поймете, что такое автомобиль, пока не узнаете, что у него внизу колеса, а посередине пустое пространство для пассажиров и двери, через которые пассажиры могут входить и выходить. То же самое относится и к белку. Чтобы понять, как он работает, вы должны точно знать, как он устроен.

Кроме того, вам нужно знать, как он взаимодействует с другими молекулами. Немногие машины работают в изоляции. Автомобиль взаимодействует с пассажирами, которых он перевозит, с дорогой, по которой он движется, и с источни-

ком энергии для движения. Это относится и к большинству белков. Они воздействуют на одни молекулы (например, чтобы ускорить химическую реакцию), на них воздействуют другие молекулы (например, регуляторы активности белка), а от третьих молекул они получают энергию. Все эти взаимодействия зависят от конкретного расположения атомов во взаимодействующих молекулах и участках белковой структуры. Чтобы расшифровать и предсказать взаимодействия с молекулами, вы должны знать, как атомы белка расположены в трехмерном пространстве.

К сожалению, вы не можете просто рассмотреть белки под микроскопом. Они слишком малы для этого. Вместо этого ученым пришлось изобрести сложные и утонченные методы исследования структуры белков. В настоящее время существует три таких метода: *рентгеновская кристаллография*, *ядерный магнитный резонанс* (ЯМР) и *криоэлектронная микроскопия* (крио-ЭМ).

Рентгеновская кристаллография является старейшим и до сих пор наиболее широко используемым методом. Примерно 90 % всех известных белковых структур были исследованы с помощью этого метода. Кристаллография начинается с выращивания кристалла белка (многие молекулы белка плотно упакованы в периодически повторяющемся порядке). Затем на кристалл направляют сфокусированные рентгеновские лучи, измеряют рассеянное излучение, а результаты анализируют, чтобы определить структуру отдельных молекул (рис. 5.1). Несмотря на широкое и успешное применение, этот метод имеет много ограничений. Он медленный и дорогой. Многие белки не образуют кристаллы, что делает кристаллографию невозможной. Кристаллизация белка иногда изменяет его структуру, поэтому результат исследования может не соответствовать структуре белка в живой клетке. Некоторые белки являются механически гибкими и могут принимать различные объемные формы, но кристаллография дает только один неподвижный снимок. Однако даже с этими ограничениями рентгеновская кристаллография остается удивительно мощным и важным инструментом.

ЯМР является вторым по распространенности методом. Он действует на белки в растворе, поэтому нет необходимости выращивать кристалл. Это делает ЯМР хорошей альтернативой для белков, которые по разным причинам не могут быть кристаллизованы. В отличие от кристаллографии, которая дает один фиксированный снимок, ЯМР регистрирует множество структур, представляющих диапазон форм, которые белок может принимать в растворе. Это очень важное преимущество, поскольку оно дает информацию о том, как молекула белка может менять свою форму. К сожалению, ЯМР имеет свои ограничения. Для этого метода требуется высококонцентрированный раствор, поэтому его применение в основном ограничено небольшими, хорошо растворимыми белками.

В последние годы появился третий способ определения структуры белка, крио-ЭМ. Молекулы белка подвергают ускоренному замораживанию при сверхнизкой температуре, а затем получают изображение структуры белка с помощью электронного микроскопа. У каждого отдельного изображения слишком низкое разрешение, чтобы разглядеть мелкие детали. Но за счет комбинации множества изображений получается окончательный снимок структуры белка с разрешением намного выше, чем у любого электронного микроскопа. После десятилетий непрерывного совершенствования методов и технологий крио-ЭМ наконец-то начала приближаться к атомному разрешению. В отличие от кристаллографии

и ЯМР, она подходит для больших белков, которые не кристаллизуются. Вероятно, в ближайшие годы крио-ЭМ станет очень важным методом исследований.

Банк белковых структур (Protein Data Bank, PDB) является основным хранилищем известных белковых структур. В настоящее время он содержит более 142 000 структур, подобных структуре, изображенной на рис. 5.1. Может показаться, что это большая коллекция, но на самом деле это намного меньше, чем действительно нужно. Количество известных белков на несколько порядков больше, и постоянно обнаруживаются новые белки. Если вы захотите изучить какой-то белок, то велика вероятность, что его структура еще не определена. Более того, для каждого белка необходимо определить несколько структур! Многие белки могут существовать в нескольких функционально разных состояниях (например, «активные» и «неактивные» состояния), поэтому вам нужно знать структуру каждого состояния. Вдобавок, если белок связывается с другими молекулами, необходимо определить структуры белка, связанного с каждой молекулой по отдельности. Эти знания помогают понять механизм возникновения связей. PDB – это прекрасный ресурс, но отрасль в целом все еще находится на стадии «малых данных». У нас гораздо меньше данных, чем хотелось бы, и главная задача – придумать, как максимально продуктивно использовать то, что есть. Это положение дел вряд ли изменится в течение ближайших десятилетий.

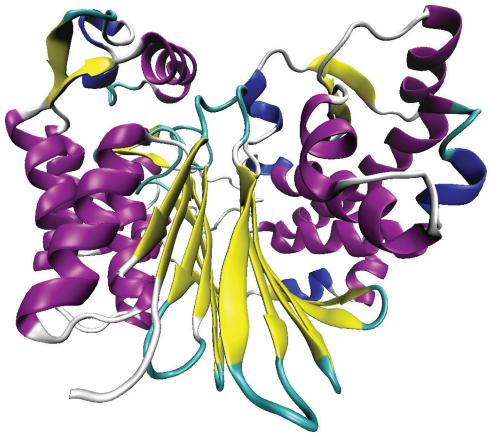


Рис. 5.1 ❖ Кристаллическая структура белка CapD из *Bacillus anthracis*, возбудителя сибирской язвы. Определение структуры бактериальных белков может стать мощным инструментом для разработки антибиотиков. В целом определение структуры *терапевтически релевантного белка* является одним из ключевых этапов в открытии современных лекарственных препаратов

Белковые последовательности

До сих пор в этой главе мы обсуждали белковые структуры, но почти не говорили о том, из чего они состоят. Белки состоят из своеобразных кирпичиков, называемых *аминокислотами* (рис. 5.2). Это наборы молекул, которые имеют общее ядро, но различные *боковые цепи* (side chain). Именно от разных боковых цепей зависит поведение белка.

Белок – это цепочка последовательно связанных аминокислот. Начало аминокислотной цепи обычно называют N-концом, в то время как окончание цепи называется С-концом (рис. 5.3). Маленькие цепочки аминокислот обычно называют *пептидами* (peptide), в то время как более длинные цепочки называют белками. Пептиды слишком малы, чтобы иметь сложные трехмерные структуры, но структуры белков бывают очень сложными.

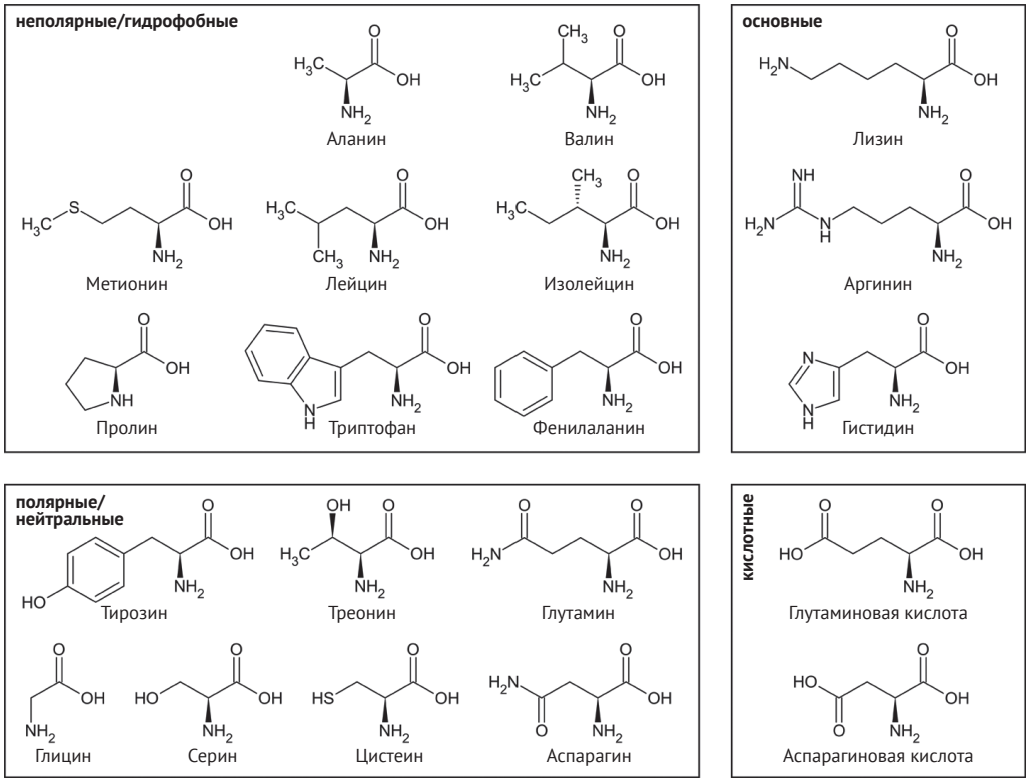


Рис. 5.2 ❖ Аминокислоты являются строительными блоками белковых структур. На этом рисунке представлена химическая структура ряда часто встречающихся аминокислот. *Источник*: **https://commons.wikimedia.org/wiki/File:Overview_proteinogenic_amino_acids-DE.svg**

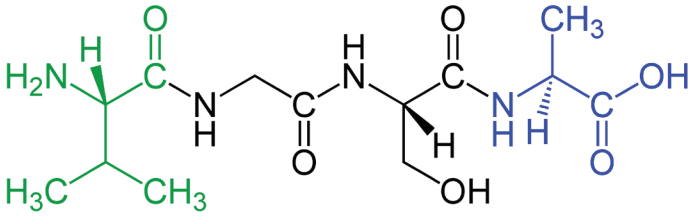


Рис. 5.3 ❖ Цепочка из четырех аминокислот, с N-концом слева и С-концом справа. *Источник*: **https://en.wikipedia.org/wiki/N-terminus#/media/File:Tetrapeptide_structural_formulae_v.1.png**

Стоит отметить, что хотя большинство белков принимает жесткую форму, существуют также *частично неупорядоченные* белки, в которых есть области, которые отказываются принимать жесткие формы (рис. 5.4).

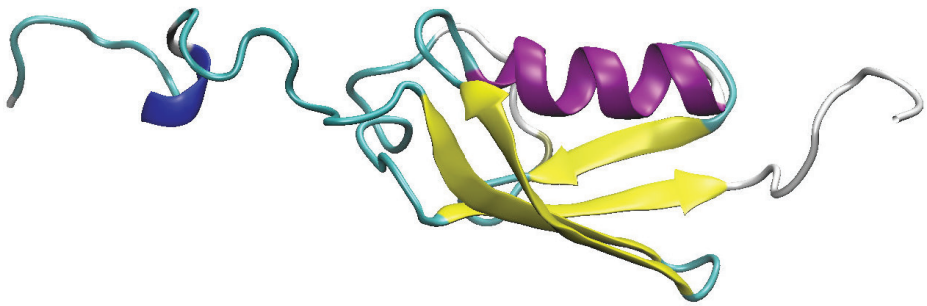


Рис. 5.4 ❖ Снимок белка SUMO-1. Центральное ядро белка имеет выраженную структуру, в то время как N-концевые и C-концевые области разупорядочены. Частично неупорядоченные белки, такие как SUMO-1, сложны в вычислительном отношении

В оставшейся части этой главы мы будем иметь дело с белками, обладающими жесткой трехмерной формой. Работа с гибкими белковыми структурами все еще является сложной задачей для современных вычислительных технологий.

Можно ли предсказать трехмерную структуру белка вычислительными методами?

Прочитав этот раздел, вы можете удивиться: почему мы не используем алгоритмы для прогнозирования структуры полезных белковых молекул, а продолжаем использовать сложнейшие физические измерения? Это хороший вопрос, и на самом деле работы по компьютерному прогнозированию белковых структур ведутся уже не одно десятилетие.

Существует два основных подхода к прогнозированию белковых структур. Первый называется *гомологичным моделированием* (homology modeling). Белковые последовательности и структуры являются продуктом миллиардов лет эволюции. Если два белка являются «близкими родственниками» (*гомологами*) и по меркам эволюции лишь недавно начали отличаться друг от друга, они, вероятно, имеют сходные структуры. Чтобы предсказать структуру белка путем гомологичного моделирования, вы сначала ищете гомолог с хорошо известной структурой, а затем пытаетесь откорректировать его на основе различий между последовательностями двух белков. Гомологичное моделирование достаточно хорошо определяет общую форму белка, но часто дает неправильные детали. И конечно же необходимо, чтобы вы достоверно знали структуру гомологичного белка.

Другой основной подход – *физическое моделирование*. Используя знания законов физики, вы перебираете множество возможных конформаций белка и пытаетесь выбрать наиболее стабильную конформацию. Этот метод требует огромных вычислительных ресурсов. Еще около десяти лет назад это было просто невозможно. Даже сегодня это приемлемо только для небольших, быстро сворачивающихся белков. Кроме того, для ускорения вычислений требуются физические

упрощения, которые снижают точность результата. Физическое моделирование предсказывает правильную структуру белка достаточно часто, но не всегда.

Общие принципы связывания с белками

До сих пор мы рассуждали о структуре белка, но мало говорили о том, как белки взаимодействуют с другими молекулами. На практике белки часто связываются с небольшими молекулами. Иногда такое поведение связывания является ключевым фактором: основная роль данного белка в организме может заключаться в связывании с определенными молекулами. Например, *сигнальная трансдукция* (signal transduction) в клетках часто передает сообщения через механизм связывания белка с другой молекулой (рис. 5.5). В некоторых случаях с белком связывается чужеродная молекула, например лекарство, которое мы создали для манипулирования белком, или токсин, нарушающий его функцию.

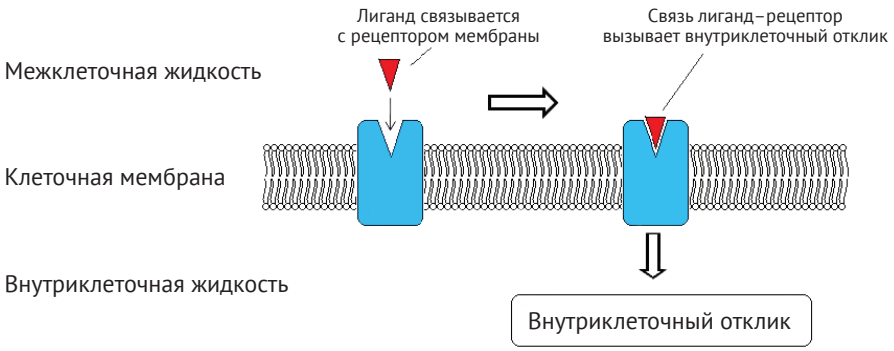


Рис. 5.5 ❖ Сигнал, передаваемый через белок, встроенный в клеточную мембрану. *Источник*: **https://simple.wikipedia.org/wiki/Signal_transduction#/media/File:The_External_Reactions_and_the_Internal_Reactions.jpg**

Разработка новых эффективных лекарств невозможна без понимания деталей того, как, где и когда молекулы связываются с белками. Воздействуя на сигнальные механизмы клеток в организме человека, мы можем вызвать желаемый *терапевтический отклик* (medical response) в организме.

Связывание с белками включает в себя множество очень специфических взаимодействий, что затрудняет вычислительное прогнозирование. Малейшее изменение в положении всего нескольких атомов может повлиять на связывание белка с молекулой. Кроме того, многие белки являются гибкими и постоянно движутся. Белок может связываться с молекулой, когда он находится в определенных конформациях, но игнорировать эту же молекулу в других конформациях. Связывание, в свою очередь, может привести к дальнейшим изменениям конформации белка и, следовательно, его функции.

В оставшейся части этой главы проблема предсказания связывания станет отправной точкой для практических вычислительных примеров. Далее мы рассмотрим современные подходы глубокого обучения и машинного обучения в прогнозировании событий связывания.

Биофизическая фичеризация

Как мы уже говорили в предыдущей главе, одним из важных шагов в применении машинного обучения к новой области является выяснение механизма *фичеризации*, то есть извлечения обучающих данных в формате, подходящем для алгоритмов обучения. Мы обсудили ряд методов фичеризации небольших молекул. Можем ли мы адаптировать эти методы для использования в биофизических системах?

К сожалению, поведение биофизических систем критически зависит от их трехмерных структур, поэтому методы для двумерных структур из предыдущих глав теряют важную информацию. В этой главе мы обсудим два новых метода фичеризации. Первый метод, *координатная фичеризация* (grid featurization), целенаправленно ищет в трехмерной структуре наличие критических физических взаимодействий, таких как *водородные связи* и *соляные мостики* (подробнее об этом позже), играющие важную роль в определении структуры белка. Преимущество этого подхода в том, что можно воспользоваться обширными знаниями о физике белков. Одновременно это и недостаток, потому что мы ограничены известными физическими зависимостями и уменьшаем вероятность того, что наши алгоритмы смогут обнаружить новые ключевые взаимодействия.

Альтернативный метод – *атомарная фичеризация* (atomic featurization) – просто дает на выходе обработанное представление трехмерных положений и идентичностей всех атомов в белковой системе. Это значительно усложняет задачу для алгоритма обучения, поскольку он должен научиться выявлять ключевые физические взаимодействия внутри системы, но, с другой стороны, позволяет алгоритму обучения обнаруживать новые шаблоны белков с интересным поведением.

Файлы PDB и их скрытые недостатки

Белковые структуры часто хранятся в *файлах PDB*. Это просто текстовые файлы, которые содержат описания атомов в структуре и их положения в координатном пространстве относительно друг друга. Алгоритмы фичеризации обычно основаны на библиотеках, которые читают файлы PDB и сохраняют их в блоках данных в памяти. Пока все хорошо, не так ли?

К сожалению, файлы PDB часто заведомо искажены. Причина заключается в физической основе белка. Зачастую экспериментальному оборудованию не хватает аппаратного разрешения, чтобы полностью исследовать часть структуры белка. Такие области не указаны в файле PDB, поэтому нередко оказывается, что в структуре отсутствуют многие атомы или даже целые подструктуры белка.

Такие библиотеки, как DeepChem, часто пытаются делать доброе дело и алгоритмически заполнять подобные недостающие области. Важно отметить, что эта правка является лишь приблизительной, и до сих пор не существует полноценной замены экспертному исследованию данных с ручной правкой неточностей. Надеемся, что в течение ближайших нескольких лет программное обеспечение для обработки данных улучшится, и потребность в экспертном вмешательстве сведется к минимуму.

Координатная фичеризация

Чтобы использовать алгоритмы машинного обучения для предсказания свойств биофизических структур, сначала нужно преобразовать эти структуры в векторы данных. Очевидно, что было бы полезно иметь алгоритм фичеризации для обработки белково-лигандных систем. Однако разработка такого алгоритма – весьма нетривиальная задача. В идеале методика фичеризации должна опираться на большой объем знаний о химическом составе таких систем и физико-химическом значении этого состава, чтобы извлекать только полезные признаки, а не все подряд.

Что это за признаки? Это могут быть нековалентные связи между белком и лигандом, такие как водородные связи или другие типы взаимодействия. (Большинство систем белок–лиганд не имеет ковалентных связей между белком и лигандом.)

К нашему счастью, у DeepChem уже есть такой набор функций, доступный в фичеризаторе RdkitGridFeaturizer. Он сводит набор соответствующей химической информации в краткий вектор для использования в алгоритмах обучения. Хотя для использования фичеризатора не нужны глубокие научные знания, все же следует познакомиться с некоторыми физическими понятиями, лежащими в основе метода.

Поэтому прежде чем углубиться в описание алгоритма координатного фичеризатора, мы сначала рассмотрим некоторые важные понятия биофизики *макромолекулярных комплексов* (macromolecular complex). При чтении этого раздела будет полезно вернуться к обсуждению основных химических взаимодействий в предыдущей главе.

Координатный фичеризатор ищет наличие таких химических взаимодействий в рамках белковой структуры и строит вектор признаков, который содержит набор этих взаимодействий. Подробнее о том, как это делается алгоритмически, мы расскажем позже в этой главе.

Водородные связи

Когда атом водорода ковалентно связан с более электроотрицательным атомом, таким как кислород или азот, общие электроны проводят большую часть своего времени ближе к более электроотрицательному атому. Атом водорода остается с суммарным положительным зарядом. Если этот положительно заряженный атом водорода затем приближается к другому атому с суммарным отрицательным зарядом, они притягиваются друг к другу. Это и есть водородная связь.

Маленькие атомы водорода могут очень близко подходить к другим атомам, что приводит к сильному электростатическому притяжению. Это делает водородные связи одним из самых сильных нековалентных взаимодействий. Они являются критически важной формой взаимодействия, которая часто стабилизирует молекулярные системы. Например, уникальные свойства воды во многом обусловлены сетью водородных связей, которые образуются между молекулами воды (рис. 5.6).

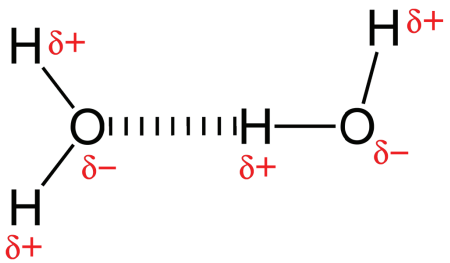


Рис. 5.6 ❖ Здесь представлен пример водородной связи. Избыточный отрицательный заряд атома кислорода взаимодействует с избыточным положительным зарядом атома водорода, создавая связующее взаимодействие. *Источник*: **https://commons.wikimedia.org/wiki/File:Hydrogen-bonding-in-water-2D.png**

RdkitGridFeaturizer пытается подсчитать водородные связи, присутствующие в структуре, путем проверки пар атомов подходящих типов, расположенных близко друг к другу. Это требует применения *расстояния отсечки* (cutoff distance), что несколько искусственно. В действительности нет четкого разделения между атомами, которые «связаны» и «не связаны». Это может привести к ошибочному определению некоторых взаимодействий, но на практике простая отсечка работает достаточно хорошо.

Соляной мостик

Соляной мостик (salt bridge) – это нековалентное притяжение между двумя аминокислотами, имеющими противоположные заряды (рис. 5.7). Мостик сочетает в себе как ионную, так и водородную связь. Хотя в данном случае эти связи относительно слабы, они помогают стабилизировать структуру белка, обеспечивая взаимодействие между удаленными аминокислотами в белковой структуре.

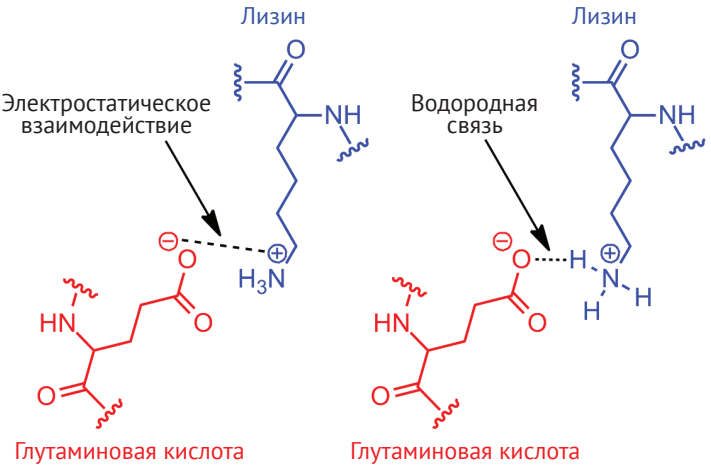


Рис. 5.7 ❖ Иллюстрация соляного мостика между глутаминовой кислотой и лизином. Соляной мостик представляет собой комбинацию электростатического взаимодействия ионного типа и водородной связи и служит для стабилизации структуры. *Источник*: **https://commons.wikimedia.org/wiki/File:Revisited_Glutamic_Acid_Lysine_salt_bridge.png**

С помощью координатной фичеризации можно попытаться обнаружить соляные мостики путем проверки пар аминокислот (например, глутаминовой кислоты и лизина как на рис. 5.7), образующих такие взаимодействия и находящихся в непосредственной физической близости в трехмерной структуре белка.

Пи-стекинг

Пи-стекинг (pi-stacking, π-stacking) является формой нековалентного взаимодействия между *ароматическими кольцами* (рис. 5.8). Это плоские, кольцеобразные структуры, которые встречаются во многих биологических молекулах, включая ДНК и РНК. Они также присутствуют в боковых цепях некоторых аминокислот, включая фенилаланин, тирозин и триптофан.

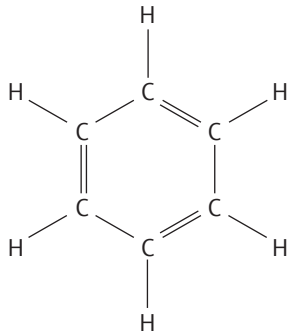


Рис. 5.8 ❖ Пример ароматического кольца в молекуле бензола. Такие кольцевые структуры известны своей исключительной стабильностью. Кроме того, все атомы ароматического кольца лежат в одной плоскости. *Гетерогенные кольца* (heterogeneous rings), напротив, не имеют атомов, занимающих одну и ту же плоскость

Грубо говоря, пи-стекинговое взаимодействие возникает, когда два ароматических кольца «укладываются» друг на друга. На рис. 5.9 показаны некоторые варианты взаимной укладки колец. Такие укладочные взаимодействия, как и солевые мостики, способствуют стабилизации макромолекулярных структур. Важно отметить, что пи-стекинг можно обнаружить в белково-лигандных связях, поскольку ароматические кольца часто встречаются в небольших молекулах. Координатный фичеризатор подсчитывает эти взаимодействия, обнаруживая наличие ароматических колец и проверяя расстояния между их центроидами и углы между их плоскостями.

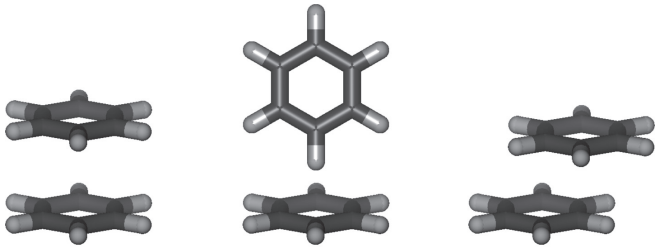


Рис. 5.9 ❖ Примеры различных нековалентных взаимодействий между ароматическими кольцами. При смещенном взаимодействии центры двух ароматических колец слегка смещены друг от друга. В перпендикулярных взаимодействиях край одного ароматического кольца накладывается на плоскость другого. Плоская конфигурация состоит из колец, сложенных непосредственно друг на друга, но энергетически менее выгодна, чем предыдущие, поскольку области с одинаковым зарядом расположены слишком близко

Наверное, вам интересно, почему этот тип взаимодействия называется пистекингом. Название относится к *пи-связям*, форме ковалентной химической свя-

зи, где электронные орбитали двух ковалентно связанных атомов перекрываются. В ароматическом кольце все атомы кольца участвуют в совместной пи-связи. Эта совместная связь обеспечивает стабильность ароматического кольца, а также объясняет многие из его уникальных химических свойств.

Для читателей, которые не являются химиками: не беспокойтесь, если этот материал сейчас кажется вам бессмысленным. DeepChem абстрагирует многие детали физической и химической реализации, поэтому при разработке моделей глубокого обучения вам не нужно будет сильно заботиться о пи-стекинге и подобных явлениях. Тем не менее полезно знать, что эти взаимодействия существуют и играют важную роль в химии.

Геометрическая путаница и мгновенные снимки

В этом разделе мы представили ряд взаимодействий в терминах *статических* геометрических конфигураций. Очень важно понимать, что связи являются *динамическими* объектами и что в реальных физических системах связи будут растягиваться, перекашиваться, рваться и видоизменяться с головокружительной скоростью. Всегда помните об этом и знайте, что если кто-то говорит, что «солевой мостик существует», он на самом деле имеет в виду, что в некотором статистическом смысле солевой мостик с большей вероятностью присутствует в указанном месте, чем в каком-либо другом.

Отпечатки

В предыдущей главе вы познакомились с понятием отпечатков молекул. Для получения отпечатка подсчитывают количество фрагментов данного типа в молекуле, а затем используют хеш-функцию для подгонки количества этих фрагментов к вектору фиксированной длины. Аналогичный метод можно применить и к трехмерным молекулярным комплексам. Хотя простого подсчета фрагментов, как правило, недостаточно для вычисления геометрии системы, знание количественной картины бывает полезным для систем машинного обучения. Возможно, это связано с тем фактом, что присутствие определенных фрагментов может достоверно указывать на некоторые молекулярные события.

Некоторые детали реализации

Для поиска химических признаков, таких как водородные связи, необходимо, чтобы `dc.feat.RdkitGridFeaturizer` мог эффективно работать с геометрией молекулы. DeepChem использует библиотеку RDKit для загрузки каждой молекулы, белка и лиганда в общий объект в памяти. Эти молекулы затем преобразовываются в массивы NumPy, которые содержат положения всех атомов в пространстве. Например, молекула с N атомами может быть представлена в виде массива NumPy формы (N, 3), где каждая строка представляет положение атома в трехмерном пространстве.

Затем для выполнения неочищенного поиска водородных связей просто рассматривают все пары атомов, которые могут образовывать водородную связь (например, кислород и водород), и отбирают такие пары, в которых атомы расположены достаточно близко друг к другу. Такая же вычислительная стратегия используется для обнаружения других видов связей. Для работы с ароматическими структурами предназначен специальный код. Он обнаруживает присутствие ароматических колец в структуре и вычисляет их центроиды.

Атомная фичеризация

В конце предыдущего раздела мы кратко рассмотрели извлечение признаков макромолекулярных структур с помощью RdkitGridFeaturizer. Молекулу с N атомами представляют в виде массива NumPy формы (N, 3), а затем выполняют множество дополнительных вычислений с использованием этих массивов.

Вы можете логично предположить, что для индивидуализации определенной молекулы достаточно лишь вычислить этот (N, 3) массив и передать его подходящему алгоритму машинного обучения. Затем модель могла бы обучиться и узнать, какие признаки важны, вместо того чтобы ожидать, что человек выберет их и закодирует вручную.

И в самом деле, это работает! Однако нужны дополнительные действия. Массив геометрических позиций (N, 3) не различает типы атомов, поэтому вам также необходимо предоставить другой массив, в котором перечислены атомные номера каждого атома. Следует заметить, что вычисление попарных расстояний между двумя позиционными массивами формы (N, 3) может быть очень дорогим в вычислительном отношении. Полезно создавать рабочие «списки соседей» на этапе предварительной обработки, где каждый список соседей содержит список соседних атомов, близких к любому данному атому.

DeepChem предоставляет набор функций `dc.feat.ComplexNeighborListFragmentAtomicCoordinates`, который выполняет для вас основную часть работы. Мы не будем углубленно изучать этот инструмент, но приятно знать, что он существует в качестве альтернативного варианта.

Пример использования PDBBind

Овладев необходимыми понятиями, перейдем к знакомству с примерами кода для обработки биофизических наборов данных. Мы начнем с изучения набора данных PDBBind и проблемы прогнозирования *свободной энергии* (free energy) связывания. Затем представим примеры кода оптимизации набора данных PDBBind и продемонстрируем, как создавать для него модели машинного обучения. Мы завершим это тематическое исследование обсуждением методики оценки результатов.

PDBBind Dataset

Набор данных PDBBind содержит большое количество *биомолекулярных* кристаллических структур и их *аффинностей связывания* (binding affinity). Давайте на минуту остановимся и разберем новые термины. *Биомолекулы* – это любые молекулы, представляющие биологический интерес. Они включают в себя не только белки, но и нуклеиновые кислоты (такие как ДНК и РНК), липиды и любые молекулы, схожие с лекарствами. Большая часть богатства биомолекулярных систем является результатом взаимодействия различных биомолекул друг с другом. Аффинность связывания – это экспериментально измеренная аффинность[1] двух молекул к образованию «комплекса», в котором две молекулы взаимодействуют между со-

[1] Взаимная степень «родства», обусловленная сочетанием различных факторов и значительно влияющая на вероятность возникновения связи и прочность связывания. – *Прим. перев.*

бой. Если энергетически выгодно формировать такой комплекс, молекулы будут проводить больше времени в этой конфигурации, а не в другой.

В наборе данных PDBBind собраны структуры ряда биомолекулярных комплексов. Подавляющее большинство из них представляет собой комплексы белок–лиганд, но набор данных также содержит комплексы белок–белок, белок – нуклеиновая кислота и нуклеиновая кислота – лиганд. Для наших целей мы сосредоточимся на подмножестве белок–лиганд. Полный набор данных содержит около 15 000 таких комплексов с «уточненными» и «основными» подмножествами, содержащими меньшие, но более чистые (проверенные и исправленные) подмножества комплексов. Каждый комплекс дополнен экспериментальными данными аффинности связывания этого комплекса. Используя набор данных PDBBind, мы должны научиться прогнозировать аффинность связывания комплекса в заданной структуре белок–лиганд.

Данные для PDBBind собираются из банка белковых структур PDB. Обратите внимание, что данные в PDB (и следовательно, в PDBBind) очень неоднородны! Разные исследовательские группы имеют разные экспериментальные установки, и результаты измерений могут существенно различаться. По этой причине для выполнения нашей экспериментальной работы мы будем в первую очередь использовать отфильтрованное уточненное подмножество PDBBind.

Динамика имеет значение!

В этом примере мы рассматриваем взаимодействие белка и лиганда как замороженный кадр. Однако этот кадр очень далек от реальной физической картины! Белок и лиганд на самом деле находятся в быстром движении, и лиганд будет постоянно входить и выходить из связывающего кармана белка. Кроме того, белок может даже не иметь одного постоянного связывающего кармана. У некоторых белков имеется несколько мест, где взаимодействуют потенциальные лиганды.

Все эти факторы означают, что наши модели будут иметь относительно ограниченную точность. Если бы у нас было больше данных, вполне возможно, что сильные обучающие модели смогли бы учитывать эти факторы, но с ограниченными наборами данных сделать это сложно. Пожалуй, вам остается лишь принять к сведению эту информацию. Разработка более совершенных биофизических моделей глубокого обучения, которые могут точно учитывать термодинамическое поведение белковых систем, остается главной открытой проблемой.

Что делать, если у вас нет структуры?

Экспериментально определить структуру комплекса обычно гораздо труднее, чем измерить аффинность связывания. Это интуитивно понятно. Аффинность связывания – это всего лишь одно число для данного биомолекулярного комплекса, в то время как структура представляет собой насыщенный трехмерный снимок. Предсказание аффинности связывания на основе структуры может немного походить на стрельбу из пушки по воробьям!

Есть несколько ответов на это, в общем-то, справедливое обвинение. Во-первых, проблема определения аффинности связывания биомолекулярной системы сама по себе является интересной задачей. Проверка способности точно предсказать аффинность связывания является достойной тестовой задачей для сравнения методов машинного обучения и может послужить отправной точкой для разработки *глубоких архитектур*, способных понимать сложные биофизические системы.

Второй ответ заключается в том, что мы можем использовать существующие вычислительные инструменты для прогнозирования приблизительных структур комплекса белок–лиганд, если знаем структуру изолированного белка. Хотя прогнозирование самой структуры белка является сложной задачей, предсказать структуру комплекса белок–лиганд на основе известной структуры белка несколько легче. Вероятно, вы сможете сделать полезные прог-

нозы с помощью системы, которую мы создадим в этом примере, если состыкуете ее с другими системами. В самом деле, DeepChem поддерживает такой вариант использования, но мы не будем углубляться в эту более продвинутую функцию в этой главе.

При машинном обучении иногда бывает полезно взглянуть на отдельные выборки данных или файлы в наборе данных. В архиве исходных кодов для этой книги мы поместили файл PDB для комплекса белок–лиганд под названием 2D3U. Он содержит информацию об аминокислотах белка, также называемых *остатками*. Кроме того, файл PDB содержит координаты каждого атома в трехмерном пространстве. Единицами измерения для этих координат являются ангстремы (1 ангстрем – 10^{-10} м). Нулевая точка системы координат выбирается произвольно, иногда из соображений лучшей визуализации, и часто устанавливается в центре тяжести белка. Мы рекомендуем потратить минуту, чтобы открыть этот файл в текстовом редакторе и посмотреть, как в нем представлены данные.

Почему аминокислота называется остатком?

Поскольку вы тратите основное время на работу с биофизическими данными, вы часто обнаруживаете, что аминокислоту называют *остатком* (residue). Этот термин указывает на химию образования белка. Когда две аминокислоты соединяются вместе в растущей цепи, то из них удаляется один атом кислорода и два атома водорода. «Остаток» – это то, что осталось от аминокислоты после этой реакции.

Понять содержимое файла PDB достаточно трудно, поэтому давайте визуализируем белок. Для наших целей мы будем использовать пакет визуализации nglview, который хорошо интегрируется с сервисом Jupyter Notebook[1]. В блокноте Jupyter, который можно найти в файловом архиве этой книги, вы сможете рассматривать визуализированный белок и взаимодействовать с ним. На рис. 5.10 представлена визуализация комплекса белок–лиганд 2D3U, сгенерированная в Jupyter Notebook.

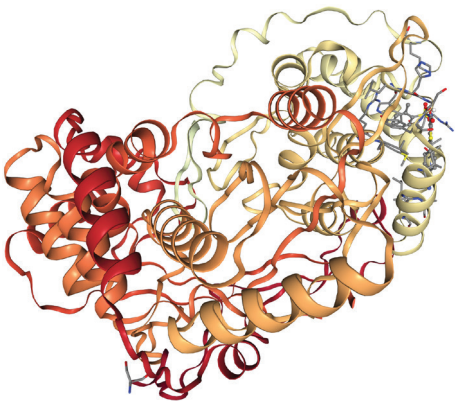


Рис. 5.10 ❖ Визуализация комплекса белок–лиганд 2D3U из набора данных PDBBind. Заметим, что белок для простоты визуализации представлен в виде ленточек, а лиганд (около верхнего правого угла) представлен в формате шарик–связь для полной детализации

[1] Онлайновый инструмент для совместной работы с документами, формулами и визуализации данных, в том числе средствами языка Python // **https://jupyter.org/**. – *Прим. перев.*

Обратите внимание, как лиганд покоится в условном «кармане» белка. Вы можете увидеть это более четко, повернув визуализацию, чтобы взглянуть на нее с разных сторон.

Инструменты визуализации белка

Визуальное представление структуры многократно повышает удобство работы с белком, поэтому нам доступно множество различных средств визуализации. Хотя пакет nglview обладает удивительной интеграцией с Jupyter, чаще встречаются другие инструменты, такие как VMD, PyMol или Chimera, используемые профессиональными исследователями лекарственных препаратов. Имейте в виду, что эти инструменты зачастую имеют закрытую лицензию и запутанный недружественный API. Тем не менее если вы планируете потратить значительное время на работу с белковыми структурами, использование одного из этих традиционных инструментов, вероятно, будет разумным компромиссом.

Представление набора данных PDBBind

Давайте начнем с создания объекта `RdkitGridFeaturizer` для проверки:

```
import deepchem as dc
grid_featurizer = dc.feat.RdkitGridFeaturizer(
    voxel_width=2.0,
    feature_types=['hbond', 'salt_bridge', 'pi_stack', 'cation_pi', 'ecfp', 'splif'],
    sanitize=True, flatten=True)
```

Здесь мы видим несколько опций, поэтому давайте остановимся и поясним их назначение. Флаг `sanitize=True` указывает фичеризатору попробовать очистить все доступные ему структуры. В примечании мы уже говорили, что структуры часто имеют неправильную или неполноценную форму. Шаг очистки будет пытаться исправить любые очевидные ошибки, которые удалось обнаружить. Флаг `feature_types` устанавливает типы биофизических и химических характеристик, которые объект `RdkitGridFeaturizer` будет пытаться обнаружить во входных структурах. Обратите внимание на знакомые химические связи, которые мы обсуждали ранее в этой главе: водородные связи, солевые мостики и т. д. Наконец, опция `voxel_width=2.0` задает размер *вокселей* (voxel), образующих пространственную сетку размером два ангстрема. `RdkitGridFeaturizer` преобразует белок в *воксельное* (voxelized) представление для использования при извлечении полезных функций. Для каждого пространственного вокселя он учитывает биофизические характеристики, а также вычисляет локальный вектор отпечатка. `RdkitGridFeaturizer` вычисляет два разных типа отпечатков – отпечатки ECFP и SPLIF.

Вокселизация

Что такое *вокселизация*? В широком смысле воксель – это трехмерный аналог пикселя (рис. 5.11). Аналогично тому, как пиксельные представления широко применяются в машинной обработке двумерных изображений, воксельные представления могут иметь решающее значение при работе с трехмерными данными.

Теперь мы готовы загрузить набор данных PDBBind. На самом деле нам не нужно использовать описанный выше фичеризатор: MoleculeNet позаботится об этом за нас. Мы можем просто включить флаг `featurizer = "grid"` при загрузке набора данных, и координатная фичеризация будет выполнена автоматически:

```
tasks, datasets, transformers = dc.molnet.load_pdbbind(
       featurizer="grid", split="random", subset="core")
train_dataset, valid_dataset, test_dataset = datasets
```

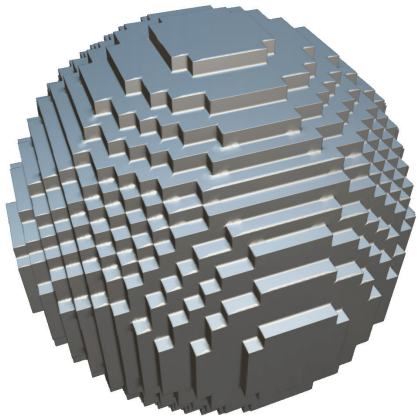


Рис. 5.11 ❖ Воксельное представление сферы. Обратите внимание, что каждый воксель представляет собой единичный «кубик» входных данных

С помощью этого фрагмента мы загрузили и фичеризовали базовое подмножество PDBBind. На быстром компьютере этот процесс займет около десяти минут. Фичеризация полного исправленного набора займет пару часов на современном сервере.

Теперь, когда у нас есть данные, давайте приступим к созданию моделей машинного обучения. Сначала мы обучим классическую модель, называемую *случайным лесом* (random forest):

```
from sklearn.ensemble import RandomForestRegressor
sklearn_model = RandomForestRegressor(n_estimators=100)
model = dc.models.SklearnModel(sklearn_model)
model.fit(train_dataset)
```

В качестве альтернативы мы также попытаемся создать нейронную сеть для прогнозирования связывания белок–лиганд. Мы воспользуемся классом `dc.models.MultitaskRegressor` для создания MLP с двумя скрытыми слоями, где установим ширину скрытых слоев на 2000 и 1000 соответственно, а также зададим исключение 50 %, чтобы уменьшить переобучение:

```
n_features = train_dataset.X.shape[1]
model = dc.models.MultitaskRegressor(
        n_tasks=len(pdbbind_tasks),
        n_features=n_features,
        layer_sizes=[2000, 1000],
        dropouts=0.5,
        learning_rate=0.0003)
model.fit(train_dataset, nb_epoch=250)
```

Базовые модели

Модели глубокого обучения не всегда легко оптимизировать. Даже опытным практикам легко ошибиться при настройке глубокой модели. По этой причине очень важно построить базовую модель, которая будет более надежной, но, возможно, будет иметь более низкую производительность.

Случайные леса очень полезны в качестве отправной точки. Как правило, они предлагают высокую производительность обучения при относительно небольших доработках. Классификатор случайных лесов строит множество классификаторов *дерева решений* (decision tree), каждый из которых использует только случайное подмножество доступных признаков, а затем объединяет отдельные решения этих классификаторов по принципу большинства голосов.

Scikit-learn – это бесценный пакет для построения простых базовых моделей машинного обучения. В этой главе мы будем применять scikit-learn для построения базовой модели, используя RdkitGridFeaturizer для фичеризации белковых комплексов в качестве входных данных для случайных лесов.

Теперь, когда у нас есть обученная модель, мы можем приступить к проверке ее точности. Чтобы оценить точность модели, мы должны сначала выбрать подходящую метрику. В данном случае будем использовать критерий Пирсона R^2. Это число от –1 до 1, где 0 указывает на отсутствие корреляции между истинной и прогнозируемой метками, а 1 указывает на идеальную корреляцию.

```
metric = dc.metrics.Metric (dc.metrics.pearson_r2_score)
```

Давайте теперь оценим точность моделей в обучающих и тестовых наборах данных в соответствии с этим показателем. Соответствующий код для этого снова передается в обе модели:

```
print("Evaluating model")
train_scores = model.evaluate(train_dataset, [metric], transformers)
test_scores = model.evaluate(test_dataset, [metric], transformers)

print("Train scores")
print(train_scores)

print("Test scores")
print(test_scores)
```

Разные архитектуры могут иметь аналогичные свойства

В этом разделе мы приводим примеры кода того, как использовать MLP с координатной фичеризацией для моделирования структур белок–лиганд в DeepChem. Важно отметить, что существует ряд альтернативных глубоких архитектур, которые имеют сходные свойства. Опубликован ряд работ по использованию трехмерных сверточных сетей для прогнозирования связывания белковых лигандов с использованием *воксельной фичеризации* (voxel-based featurization). В другой работе для обработки макромолекулярных комплексов использовались варианты графовых сверток, которые мы обсуждали в предыдущих главах. Каковы различия между этими архитектурами? Пока похоже, что большинство из них имеют схожую предсказательную силу. Мы используем координатную фичеризацию, поскольку в DeepChem есть готовая настроенная реализация этого подхода, но и другие модели вполне могут удовлетворить ваши потребности. В будущие версии DeepChem, вероятно, будут добавлены и другие архитектуры.

Для случайного леса оценка обучающего набора составляет 0,979, а оценка тестового набора – только 0,133. Он отлично справляется с воспроизведением обучающих данных, но очень плохо предсказывает тестовые данные. Очевидно, что это слишком плохой результат.

Для сравнения, нейронная сеть имеет оценку на обучающем наборе 0,990 и оценку на тестовом наборе 0,359. Это немного лучше на тренировочном наборе

и заметно лучше на тестовом наборе. Очевидно, что все еще происходит переобучение, но степень переобучения уменьшается, и общая способность модели прогнозировать новые данные намного выше.

Знание коэффициента корреляции является мощным первым шагом к пониманию построенной нами модели, но всегда полезно визуализировать, как наши прогнозы соотносятся с фактическими экспериментальными данными. На рис. 5.12 показаны метки фактического и прогнозного значения для каждой из моделей при запуске на тестовом наборе. Мы сразу видим, что предсказания нейронной сети гораздо более тесно связаны с фактическими данными, чем прогнозы случайного леса.

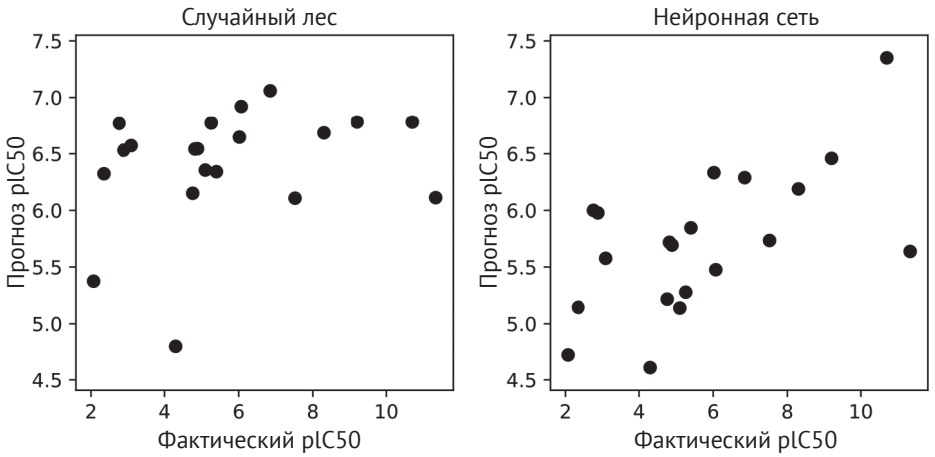


Рис. 5.12 ❖ Метки фактических и прогнозных значений для двух моделей после запуска на проверочном наборе

Заключение

В этой главе вы узнали о применении глубокого обучения к биофизическим системам и, в частности, к проблеме прогнозирования аффинности связывания белково-лигандных систем. Вам может быть интересно, насколько общими являются приобретенные вами навыки. Можно ли применить те же модели и методы для изучения других биофизических наборов данных? Давайте кратко обсудим этот вопрос.

Системы *белок–белок* и *белок–ДНК* на высоком уровне следуют той же базовой физике, что и белково-лигандные системы. Критическую роль играют те же водородные связи, солевые мостики, пи-стекинговые взаимодействия и т. д. Можем ли мы просто повторно использовать код из этой главы для анализа таких систем? Ответ немного сложен и требует пояснений. Многие из физических взаимодействий, которые управляют связью белок–лиганд, определяются динамикой электростатических зарядов. С другой стороны, динамика связи белок–белок по большей части может быть обусловлена гидрофобными взаимодействиями. Мы не будем углубляться в смысл этих взаимодействий, но они в какой-то степени имеют качественно иной характер, чем белково-лигандные взаимодействия. Это

означает, что `RdkitGridFeaturizer` не сможет хорошо охарактеризовать эти взаимодействия. С другой стороны, есть надежда, что атомные сверточные модели смогут лучше справляться с системами белок–белок, поскольку эти глубокие модели гораздо меньше опираются на физику взаимодействий.

Тем не менее остается серьезная проблема масштабных вычислений. Атомные сверточные модели все еще довольно медленны в обучении и требуют много памяти. Масштабирование этих моделей для обработки больших белково-белковых систем потребует дополнительной работы на инженерном уровне. Разработчики DeepChem усердно работают над этой и другими проблемами, но может пройти какое-то время, прежде чем эти усилия дадут свои результаты.

Взаимодействия *антитело–антиген* являются еще одной формой критически важного биофизического взаимодействия. *Антитела* (antibody) представляют собой Y-образные белки (рис. 5.13), которые имеют специальные *участки связывания антигена* (antigen-binding site). *Антигены* (antigen) представляют собой молекулы, связанные с конкретным патогеном. В последнее время антитела получили большее применение в терапевтической области благодаря использованию так называемых *моноклональных антител*. Искусственно выращенную культуру клеток (подробнее об этом в следующей главе) можно использовать для создания антител, которые нацелены на специфические антигены. Если все клетки в культуре являются клонами одной исходной клетки, то полученные антитела будут идентичны, т. е. моноклональны. Такие моноклональные антитела с недавнего времени широко применяются в медицине.

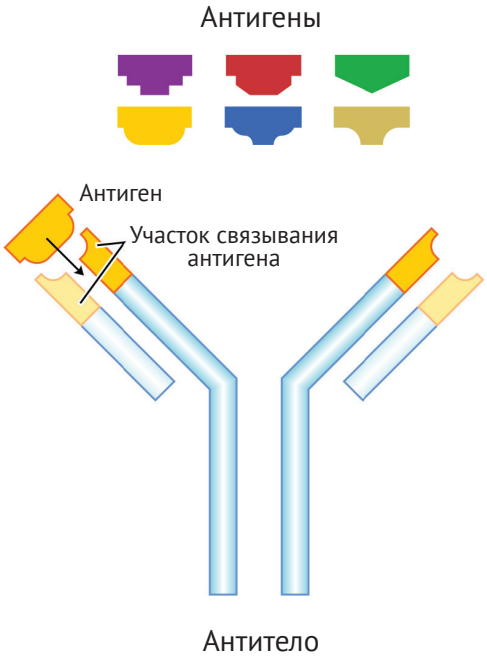


Рис. 5.13 ❖ Схема взаимодействия антитело–антиген.
Источник: **https://en.wikipedia.org/wiki/Antibody#/media/File:Antibody.svg**

Разработка антител до сих пор была исключительно экспериментальной наукой. Частично это связано с трудностью получения трехмерной структуры антител. Однако моделирование участка связывания антигена также оказалось сложной задачей. Некоторые из методов, которые мы рассмотрели в этой главе, могут найти плодотворное применение в моделировании связи антитело–антиген в течение следующих нескольких лет.

Мы также говорили о роли динамических процессов в физике белка. Можно ли на основе глубокой модели построить компьютерную симуляцию белка, чтобы понять, какие лиганды могут связываться с белком? В принципе, да, но остаются серьезные проблемы. Некоторые компании активно работают над этим вопросом, но готовых инструментов с открытым исходным кодом пока нет.

В главе 11 мы вернемся к биофизическим методам и покажем, что эти модели могут быть очень полезны при разработке новых лекарств.

Глава 6

Глубокое обучение и геномика

В основе каждого живого организма лежит его *геном* – молекулы *ДНК*, содержащие полный набор инструкций для создания рабочих частей организма. Если клетка – это компьютер, то последовательность ее генома – это программа, которую она выполняет. И если ДНК можно рассматривать как программное обеспечение или информацию, предназначенную для обработки компьютером, то, наверное, мы можем использовать компьютеры для анализа этой информации и расшифровки ее назначения?

Но, конечно же, ДНК – это не просто абстрактное хранилище информации. Это физическая молекула, которая ведет себя сложным образом. Она взаимодействует с тысячами других молекул, играющих важную роль в поддержании, копировании и выполнении инструкций, содержащихся в ДНК. Геном – это огромная и сложная машина, состоящая из тысяч частей. Мы до сих пор плохо понимаем, как работает большинство этих частей, не говоря уже о том, как все они образуют единое целое.

Эти рассуждения подводят нас к двойному понятию генетики и геномики. *Генетика* рассматривает ДНК как абстрактную информацию. Она изучает паттерны наследования или ищет корреляции между популяциями организмов, чтобы обнаружить связи между последовательностями ДНК и особенностями организма. *Геномика*, с другой стороны, рассматривает геном как физическую машину, пытаясь понять части, составляющие эту машину, и принципы их взаимодействия. Два подхода дополняют друг друга, и глубокое обучение может стать мощным инструментом для них обоих.

ДНК, РНК и белки

Даже если вы не биолог, в средней школе вы наверняка изучали основные принципы работы генома. Сначала мы вспомним упрощенную схему геномики, которая обычно преподается школьникам. Затем мы поговорим о том, что реальные генетические механизмы устроены намного сложнее.

Молекула ДНК – это *полимер*, то есть длинная цепь повторяющихся звеньев. В случае ДНК различают четыре типа звеньев, так называемые *основания*: аденин, цитозин, гуанин и тимин, которые сокращенно обозначаются как A, C, G и T (рис. 6.1). Почти вся информация о живом организме в конечном итоге закодиро-

вана в специфической последовательности этих четырех повторяющихся оснований, составляющих его геном.

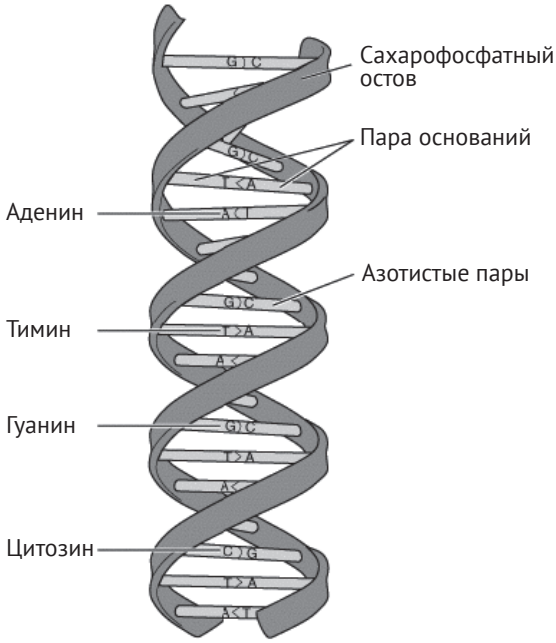


Рис. 6.1 ❖ Структура молекулы ДНК. Она состоит из двух цепей, каждая из которых составлена из множества оснований A, C, G и T. Две цепочки являются комплементарными: каждое основание C в одной цепочке связано с G в другой, а каждое основание A в одной цепочке связано с T в другой. *Источник*: **https://en.wikipedia.org/wiki/Molecular_Structure_of_Nucleic_Acids:_A_Structure_for_Deoxyribose_Nucleic_Acid#/media/File:DNA-structure-and-bases.png**

Если считать ДНК программой, то белки являются наиболее важным вычислительным оборудованием. Белки – это крошечные машины, выполняющие почти всю работу в клетке. Они тоже являются полимерами и состоят из повторяющихся звеньев, называемых *аминокислотами*. Есть 20 основных белковых аминокислот с очень разными физическими свойствами. Одни большие, а другие маленькие. Одни имеют электрический заряд, а другие нет. Одни склонны притягивать воду, а другие – отталкивают ее. Как только нужный набор аминокислот соединяется в правильном порядке, он самопроизвольно складывается в трехмерную форму, где все части располагаются именно так, чтобы белок функционировал как машина.

Одна из основных функций ДНК заключается в записи последовательности аминокислот для белков организма. Это делается простым и понятным способом. Конкретные участки ДНК напрямую соответствуют конкретным белкам. Каждые три основания ДНК (называемых *кодонами*) соответствуют одной аминокислоте. Например, шаблон AAA указывает на аминокислоту лизин, тогда как шаблон GCC указывает на аминокислоту аланин.

Перенос от ДНК к белку происходит через молекулу *РНК*, выступающую посредником для передачи информации из одной части клетки в другую. Молекула РНК является еще одним полимером и химически очень похожа на ДНК. У нее тоже есть четыре основания, которые могут соединяться в цепи в произвольном порядке. При создании белка происходит двукратное копирование информации. Сначала последовательность ДНК *транскрибируется* (transcribe) в эквивалентную последовательность РНК, а затем последовательность РНК *транслируется* (translate) в молекулу белка. Молекула РНК, которая несет информацию, называется *матричной*, или *информационной*, РНК (messenger RNA), или сокращенно мРНК (иРНК).

Это описание говорит нам, *как* получаются белки, но не поясняет, *когда* они получаются. Клетка человека содержит много тысяч различных белков, которые необходимо когда-то произвести. Но клетка ведь не вырабатывает копии всех белков подряд и без остановки? Ясно, что должен быть какой-то регуляторный механизм, контролирующий производство белков. В обычном клеточном механизме это делается с помощью специальных белков, называемых *факторами транскрипции* (transcription factor, TF). Каждый фактор транскрипции связывается с определенной последовательностью ДНК. В зависимости от конкретного фактора транскрипции и фрагмента ДНК, с которым он связан, он может увеличивать или уменьшать скорость, с которой транскрибируются близлежащие гены.

Все вместе дает простую и понятную картину того, как работает геном. Работа ДНК заключается в кодировании белков. Участки ДНК (называемые *генами*) кодируют белки с использованием простого, четко определенного кода. ДНК отображается в РНК, которая служит только переносчиком информации. Затем РНК конвертируется в белки, которые выполняют в клетке всю текущую работу. Чрезвычайно элегантный процесс, который мог бы придумать гениальный инженер. И на протяжении многих лет эта картина считалась в целом правильной и безупречной. Так что остановитесь на какое-то время и насладитесь красотой мироздания, прежде чем мы испортим вам настроение, показав, что реальность на самом деле намного сложнее и запутаннее.

Реальное положение дел

Теперь пришло время поговорить о том, как на самом деле работают геномы. Картина, описанная выше, проста и элегантна, но, к сожалению, она мало связана с реальностью. В этом разделе будет очень много новой информации, но не беспокойтесь о том, чтобы запомнить или понять ее с первого раза. Главное – ощутить невероятную сложность живых организмов. Мы вернемся к некоторым темам этого раздела позже и обсудим их более подробно.

Давайте начнем с рассмотрения молекул ДНК, называемых *хромосомами*. У бактерий, имеющих относительно небольшие геномы, ДНК представлена в виде простых свободно плавающих молекул. Однако у *эукариот* (eucaryotes, все организмы, от амеб до людей, у которых клетки имеют выраженное ядро) геномы гораздо больше. Чтобы поместиться внутри клетки, каждая хромосома должна быть упакована в очень маленьком объеме. Это достигается путем наматывания хромосомы вокруг белков, называемых *гистонами* (histones). Но если вся ДНК так плотно упакована, как ее можно транскрибировать? Разумеется, никак. Прежде

чем ген можно будет транскрибировать, необходимо размотать нужный участок ДНК! Откуда клетка знает, какой участок ДНК размотать? На этот вопрос до сих пор нет внятного ответа. Считается, что это как-то связано с различными химическими модификациями молекул гистона и белками, которые распознают определенные модификации. Очевидно, что существует регулирующий механизм, но многие детали его работы до сих пор неизвестны. Мы вернемся к этой теме в ближайшее время.

Сама ДНК тоже может быть химически модифицирована, этот процесс называется *метилированием* (methylation). Чем сильнее метилирован участок ДНК, тем меньше вероятность его транскрибирования, так что это еще один регуляторный механизм, который клетка может использовать для управления производством белков. Но от чего зависит метилирование определенных участков ДНК? На этот вопрос тоже нет окончательного ответа.

В предыдущем разделе мы говорили, что определенный участок ДНК соответствует определенному белку. Это верно для бактерий, но в случае эукариот ситуация намного сложнее. После того как ДНК транскрибируется в РНК-переносчик, эта РНК часто редактируется путем удаления секций и последующего соединения, так называемого *сплайсинга* (splice) оставшихся частей, называемых *экзонами* (exon). Выходит, что последовательность РНК, которая в конечном итоге транслируется в белок, может отличаться от исходной последовательности ДНК. Кроме того, многие гены имеют несколько вариантов сплайсинга, то есть различные способы удаления секций для формирования конечной последовательности РНК. Это означает, что один участок ДНК может кодировать несколько разных белков!

Все это звучит очень сложно? Да, это так, но вам придется напрячь внимание. Эволюция выбирает наиболее эффективные механизмы, не заботясь об их простоте. Она порождает очень сложные системы, и для понимания устройства этих систем необходимо преодолеть сложности.

В упрощенном представлении клетки РНК рассматриваются как простой носитель информации, но даже с первых дней геномики биологи знали, что это не совсем правильно. Работа по трансляции мРНК в белки выполняется *рибосомами* (ribosomes), сложными молекулярными машинами, сделанными частично из белков и частично из РНК. Другую ключевую роль в трансляции играют молекулы, называемые *транспортными РНК* (transfer RNA, тРНК). Это молекулы, которые определяют «генетический код», распознавая шаблоны трех оснований в мРНК и добавляя правильную аминокислоту к растущему белку. Таким образом, биологи знали, что существуют по крайней мере три вида РНК: мРНК, рибосомная РНК и тРНК.

Но у РНК все еще припасены тузы в рукаве. Это удивительно универсальная молекула. За последние десятилетия было открыто много новых типов РНК. Вот несколько примеров:

- *микроРНК* (micro RNA, miRNA) представляют собой короткие фрагменты РНК, которые связываются с РНК-переносчиком и предотвращают трансляцию в белки. Это очень важный регуляторный механизм у некоторых видов животных, особенно у млекопитающих;
- *короткая интерферирующая РНК* (short interfering RNA, siRNA) – это другой тип РНК, который связывается с мРНК и препятствует ее трансляции. Они похожи на микроРНК, но являются двухцепочечными и отличаются некото-

рыми деталями функционирования. Мы более подробно обсудим короткие РНК позже в этой главе;

- *рибозимы* – это молекулы РНК, которые могут выступать в роли ферментов, катализирующих химические реакции. Химия является основой всего, что происходит в живой клетке, поэтому клеточные катализаторы в буквальном смысле жизненно важны. Обычно эту работу выполняют белки, но теперь мы знаем, что иногда это делается с помощью РНК;
- *рибосвитчи* (riboswitches, РНК-переключатели) – это молекулы РНК, которые состоят из двух частей. Одна часть действует как РНК-переносчик, а другая часть способна связываться с небольшой молекулой. Наличие связи с молекулой может включить или, наоборот, выключить транскрипцию мРНК. Это еще один регуляторный механизм, с помощью которого можно управлять выработкой белка за счет изменения концентрации определенных малых молекул в клетке.

Разумеется, все эти различные типы РНК должны быть произведены заранее, и ДНК должна содержать инструкции о том, как это сделать. Таким образом, ДНК – это больше, чем просто набор кодированных белковых последовательностей. Она также содержит последовательности РНК и *сайты связывания* (binding sites) для факторов транскрипции и других регуляторных молекул, плюс инструкции о том, как следует сплайсировать транспортную РНК, а также различные химические модификации, влияющие на то, как ДНК наматывается на гистоны и какие гены транскрибируются.

Теперь рассмотрим, что происходит после того, как рибосома заканчивает трансляцию мРНК в белок. Некоторые белки могут самопроизвольно складываться в правильную трехмерную форму, но многим из них нужна помощь других белков, называемых *шаперонами* (chaperone). После трансляции белки часто нуждаются в дополнительной химической модификации. Затем готовый белок должен как-то попасть в нужное место в клетке, чтобы выполнить свою работу, и наконец разложиться, когда он больше не нужен. Каждый из этих процессов управляется дополнительными регуляторными механизмами и включает взаимодействие со множеством других молекул.

Все это звучит ошеломляюще, но иначе и быть не может. Живой организм намного сложнее, чем любая машина, созданная людьми. Даже мысль о попытке понять устройство организма должна вызывать у вас трепет!

Но именно поэтому машинное обучение является таким мощным инструментом. У нас есть огромные объемы данных, сгенерированных процессами, которые невероятно сложны и плохо поняты. Мы хотим обнаружить тонкие закономерности, скрытые в данных. Это как раз та проблема, для которой предназначено глубокое обучение.

На самом деле глубокое обучение *уникально* подходит для этой проблемы. Классические статистические методы изо всех сил пытаются математически представить сложнейшее устройство генома. Они часто основаны на упрощении предположений. Например, они ищут линейные отношения между переменными или пытаются смоделировать зависимость переменной только от небольшого числа других переменных. Но геномика включает в себя сложные нелинейные отношения между сотнями переменных – именно такие отношения могут быть эффективно описаны глубокой нейронной сетью.

Сайты связывания и факторы транскрипции

В качестве примера применения глубокого обучения к геномике давайте рассмотрим прогнозирование связывания факторов транскрипции. Напомним, что *факторы транскрипции* (TF) представляют собой белки, которые связываются с ДНК и влияют на вероятность транскрибирования соседних генов в РНК. Но как TF находит нужный центр связывания? Как свойственно геномике, на этот вопрос есть простой ответ, сопровождаемый множеством усложнений.

В первом приближении каждому TF соответствует специфическая последовательность ДНК, называемая *мотивом сайта связывания* (binding site motif). Мотивы сайтов связывания – это небольшие участки, обычно не более 10 оснований. TF связываются с ДНК в местах расположения соответствующих мотивов.

Однако на практике мотивы не совсем специфичны. TF может связываться со многими похожими, но не идентичными последовательностями. Некоторые основания в мотиве могут быть важнее других. При моделировании часто применяется *позиционная весовая матрица* (position weight matrix), которая указывает, какое значение для TF имеет каждое возможное основание в каждой позиции в мотиве. Конечно, это предполагает, что каждая позиция в мотиве независима, что не всегда верно. Иногда может варьироваться даже длина мотива. И хотя связывание в первую очередь определяется основаниями внутри мотива, ДНК по обе стороны от него также может иметь некоторое влияние.

И это мы говорили только о фрагментах последовательности ДНК! Другие аспекты ДНК могут быть не менее важны. На многие TF влияет физическая форма ДНК, например насколько сильно закручена двойная спираль. Если ДНК метилирована, это может повлиять на связывание TF. И помните, что большая часть ДНК у эукариот плотно упакована и обмотана вокруг гистонов. TF могут связываться только с теми частями, которые были размотаны.

Другие молекулы также играют важную роль. TF часто взаимодействуют с другими молекулами, и эти взаимодействия могут влиять на связывание с ДНК. Например, TF может связаться с другой молекулой и образовать комплекс. Но этот комплекс будет склонен связываться с другим мотивом ДНК, отличным от того, с которым связывался исходный TF.

Биологи потратили десятилетия, разбирая эти детали и разрабатывая модели связывания TF. Вместо того чтобы повторять их подход, давайте посмотрим, сможем ли мы использовать глубокое обучение, чтобы получить модель непосредственно из данных.

Сверточная модель связывания TF

Для этого примера мы будем использовать экспериментальные данные о конкретном факторе транскрипции, называемом JUND. Для определения расположения каждого сайта связывания JUND в геноме человека была проведена серия экспериментов. Для наглядности мы воспользуемся только данными для 22-й хромосомы, одной из самых маленьких человеческих хромосом. Ее длина составляет более 50 млн оснований, что дает нам достаточное количество данных для работы. Полная хромосома была разделена на короткие сегменты из 101 основания, и каждый сегмент пометили, чтобы указать, содержит ли он сайт связыва-

ния JUND. Мы попытаемся обучить модель, которая предсказывает эти метки на основе последовательности каждого сегмента.

Последовательности оснований представлены *унитарным кодом* с одним активным состоянием. Для каждого основания у нас есть четыре числа, из которых только одно установлено в 1, а остальные в 0. Значение 1 для соответствующего числа указывает на основание A, C, G или T.

Для обработки данных мы будем использовать сверточную нейронную сеть, как мы это делали для распознавания рукописных цифр в главе 3. На самом деле, как вы увидите, две модели удивительно похожи друг на друга. Конечно, в этот раз мы будем использовать одномерные свертки, поскольку имеем дело с одномерными данными (последовательностями оснований) вместо двумерных данных (изображений). Но основные компоненты модели останутся прежними: входные данные, ряд сверточных слоев, один или несколько *полносвязных слоев* (dense layer) для вычисления выходных данных и *кросс-энтропийная функция потерь* (cross entropy loss function).

Давайте начнем с создания экземпляра объекта TensorGraph и определения входных данных:

```
model = dc.models.TensorGraph(batch_size=1000)
features = layers.Feature(shape=(None, 101, 4))
labels = layers.Label(shape=(None, 1))
weights = layers.Weights(shape=(None, 1))
```

Обратите внимание на размеры входов. Для каждого образца у нас есть вектор признаков размером 101 (количество оснований) на 4 (длина унитарного кода основания). У нас также есть одно число для метки (0 или 1, чтобы указать, содержит ли образец сайт связывания) и одно число для веса. Для этого примера имеет решающее значение использование весов в функции потерь, потому что данные очень сильно разбалансированы. Сайт связывания содержит менее 1 % всех выборок. Это означает, что модель может получить точность, превышающую 99 %, просто выдав 0 для каждой выборки. Мы предотвращаем это, давая положительным выборкам более высокий вес, чем отрицательным.

Далее мы создаем стек из трех сверточных слоев, все с одинаковыми параметрами:

```
prev = features
for i in range(3):
    prev = layers.Conv1D(filters=15, kernel_size=10,
                         activation=tf.nn.relu, padding='same',
                         in_layers=prev)
prev = layers.Dropout(dropout_prob=0.5, in_layers=prev)
```

Мы указываем, что ширина сверточных ядер равна 10 и что каждый слой должен включать 15 фильтров (то есть выходов). Первый слой принимает исходные объекты (4 числа на основание) в качестве входных данных. Он рассматривает промежутки из 10 последовательных оснований, итого 40 входных значений. Для каждого промежутка он умножает эти 40 значений на сверточное ядро, чтобы получить 15 выходов. Второй слой снова смотрит на промежутки из 10 оснований, но на этот раз входными данными являются 15 значений, вычисленных

первым слоем. Он вычисляет новый набор из 15 значений для каждого основания и т. д.

Чтобы предотвратить переобучение, после каждого сверточного слоя мы добавляем *слой исключений* (droput layer). Вероятность исключения установлена на 0,5, что означает случайное обнуление 50 % всех выходных значений.

Затем мы используем полносвязный слой для вычисления результата:

```
logits = layers.Dense(out_channels=1, in_layers=layers.Flatten(prev))
output = layers.Sigmoid(logits)
model.add_output(output)
```

Мы хотим, чтобы выходные значения лежали в диапазоне между 0 и 1, чтобы интерпретировать их как вероятность наличия сайта связывания в конкретном образце. Полносвязный слой может давать произвольные значения, не ограниченные каким-либо конкретным диапазоном. Поэтому мы пропускаем его выход через *логистическую сигмоидную функцию* (logistic sigmoid function), чтобы сжать до желаемого диапазона. Входные значения для этой функции часто называют *логитами* (logit). Название относится к математической *логит-функции*, которая является обратной функцией логистической сигмоиды.

Наконец, мы вычисляем перекрестную энтропию для каждой выборки и умножаем на веса, чтобы получить потери:

```
loss = layers.SigmoidCrossEntropy(in_layers=[labels, logits])
weighted_loss = layers.WeightedError(in_layers=[loss, weights])
model.set_loss(weighted_loss)
```

Обратите внимание, что для обеспечения численной устойчивости кросс-энтропийный слой в качестве входных данных принимает логиты, а не выходные данные сигмоидальной функции.

Теперь мы готовы обучить и оценить модель. В качестве метрики оценки мы используем ROC-AUC. После каждых 10 эпох обучения мы оцениваем модель как на обучающем, так и на проверочном наборе.

```
train = dc.data.DiskDataset('train_dataset')
valid = dc.data.DiskDataset('valid_dataset')
metric = dc.metrics.Metric(dc.metrics.roc_auc_score)
for i in range(20):
    model.fit(train, nb_epoch=10)
    print(model.evaluate(train, [metric]))
    print(model.evaluate(valid, [metric]))
```

Результат показан на рис. 6.2. Оценка качества на проверочном наборе достигает пика около 0,75 после 50 эпох, а затем немного уменьшается. Оценка на обучающем наборе продолжает расти и в конечном итоге выравнивается на отметке 0,87. Это говорит нам о том, что тренировки за пределами 50 эпох просто приводят к переобучению и в этой точке мы должны остановиться.

Оценка ROC-AUC 0,75 не плохая, но и не замечательная. Возможно, мы могли бы увеличить ее, улучшив модель. Существует множество доступных для настройки гиперпараметров: количество сверточных слоев, ширина ядра для каждого слоя, количество фильтров в каждом слое, частота исключения и т. д. Мы могли бы попробовать множество комбинаций гиперпараметров и выбрать из них наилучшее сочетание.

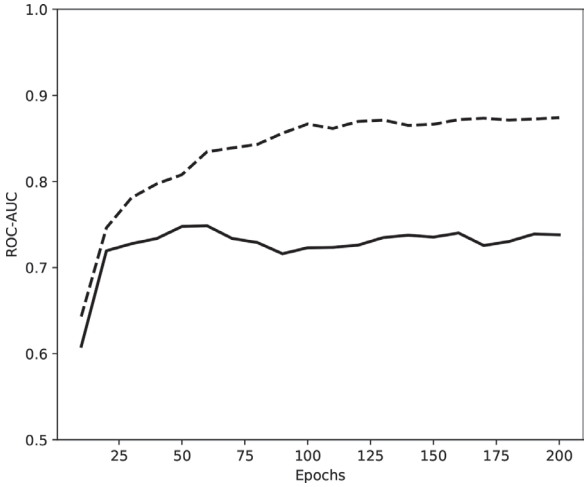


Рис. 6.2 ❖ Эволюция ROC-AUC во время обучения для обучающего набора (пунктир) и проверочного набора (сплошная линия)

Но мы также знаем, что существуют фундаментальные ограничения модели. Единственный вход, с которым она работает, – это последовательность ДНК, хотя связывание TF также зависит от множества других факторов: доступности, метилирования, формы, присутствия других молекул и т. п. Любая модель, которая игнорирует эти факторы, будет ограничена в точности предсказаний. А теперь давайте попробуем добавить второй вход и посмотрим, поможет ли это.

Доступность хроматина

Название *хроматин* (chromatine) относится ко всему, что составляет хромосому: ДНК, гистоны и другие белки и молекулы РНК. *Доступность хроматина* (chromatine accessibility) относится к тому, насколько доступна каждая часть хромосомы для внешних молекул. Когда ДНК плотно обвивается вокруг гистонов, она становится недоступной для факторов транскрипции и других молекул. Когда ДНК разматывается с гистонов, она снова становится доступной и продолжает играть роль центральной части генетического механизма клетки.

Доступность хроматина не является ни равномерной, ни статичной. Она меняется в зависимости от типа и стадии жизненного цикла клетки. На доступность могут повлиять условия окружающей среды. Это один из механизмов регулирования активности генома. Любой ген можно отключить, упаковав участок хромосомы, где он находится.

Доступность также постоянно меняется, поскольку ДНК сматывается и разматывается в ответ на события в клетке. Вместо того чтобы думать о доступности как о бинарном выборе (доступен/недоступен), лучше думать о ней как о непрерывной переменной (какую часть времени доступен определенный регион).

Данные, которые мы проанализировали в предыдущем разделе, были получены в результате экспериментов с клетками определенного типа, так называемыми HepG2. Напомним, что места связывания фактора транскрипции JUND определены экспериментально. На результаты связывания оказала влияние доступность

хроматина. Если определенная область в клетках HepG2 почти всегда недоступна, эксперимент вряд ли обнаружит в этом месте связывание JUND, даже если последовательность ДНК идеально подходит на роль сайта связывания. Итак, давайте попробуем включить доступность хроматина в нашу модель.

Сначала загрузим набор данных о доступности. У нас есть данные в виде текстового файла, где каждая строка соответствует одной выборке из нашего набора данных (101 базовый отрезок 22-й хромосомы). Строка содержит идентификатор выборки, за которым следует число, которое показывает, насколько доступен этот регион в клетках HepG2. Мы загрузим файл в словарь Python:

```
span_accessibility = {}
for line in open('accessibility.txt'):
    fields = line.split()
    span_accessibility[fields[0]] = float(fields[1])
```

Теперь для построения модели мы будем использовать практически ту же модель, что и в предыдущем разделе, но с двумя небольшими изменениями. Во-первых, нам нужен вход второго признака – значений доступности хроматина. Это одно число для каждого образца.

```
accessibility = layers.Feature(shape=(None, 1))
```

Теперь нам нужно включить в расчеты значение доступности. Это можно сделать разными способами. В данном примере мы будем использовать самый простой метод. В предыдущем разделе мы сгладили выходные данные последнего сверточного слоя, а затем использовали его в качестве входных данных для полносвязного слоя, который вычислял выходные данные:

```
logits = layers.Dense(out_channels=1, in_layers=layers.Flatten(prev))
```

На этот раз мы сделаем то же самое, но также добавим к выводу свертки доступность:

```
prev = layers.Concat([layers.Flatten(prev), accessibility])
logits = layers.Dense(out_channels=1, in_layers=prev)
```

Вот и вся модель! Теперь пришло время обучить ее.

На этом этапе мы столкнулись с трудностью: наша модель имеет два разных слоя Feature. До сих пор наши модели имели ровно один векторный слой, один слой меток и, возможно, один слой весов. Мы обучили их, вызвав метод `fit(dataset)`, который автоматически связал правильные данные с каждым слоем – поле **X** набора данных для признаков, **y** для меток и **w** для весов. Но такой подход явно не может работать, когда модель имеет более одного набора признаков.

Эта ситуация решается с помощью более продвинутых возможностей DeepChem. Вместо того чтобы передавать набор данных в модель, мы можем написать функцию Python, которая перебирает пакеты. Каждый пакет представлен словарем, ключи которого являются входными слоями, а значения – это массивы NumPy, которые будут использоваться для них.

```
def generate_batches(dataset, epochs):
    for epoch in range(epochs):
        for X, y, w, ids in dataset.iterbatches(batch_size=1000, pad_batches=True):
            yield {
                features: X,
```

```
        accessibility: np.array([span_accessibility[id] for id in ids]),
        labels: y,
        weights: w
    }
```

Обратите внимание на то, как `dataset` заботится о переборе пакетов. Он предоставляет данные для каждой партии, из которых мы можем построить любые входные данные, необходимые для модели.

Обучение и оценка происходят точно так же, как и раньше. Мы используем альтернативные формы методов, которые вместо набора данных используют генератор.

```
for i in range(20):
model.fit_generator(generate_batches(train, 10))
print(model.evaluate_generator(generate_batches(train, 1), [metric],
                               labels=[labels], weights=[weights]))
print(model.evaluate_generator(generate_batches(valid, 1), [metric],
                               labels=[labels], weights=[weights]))
```

Результат показан на рис. 6.3. Оценки как на обучающем, так и на проверочном наборе стали лучше по сравнению с моделью, игнорирующей доступность хроматина. ROC-AUC теперь достигает 0,91 для обучающего набора и 0,80 для проверочного набора.

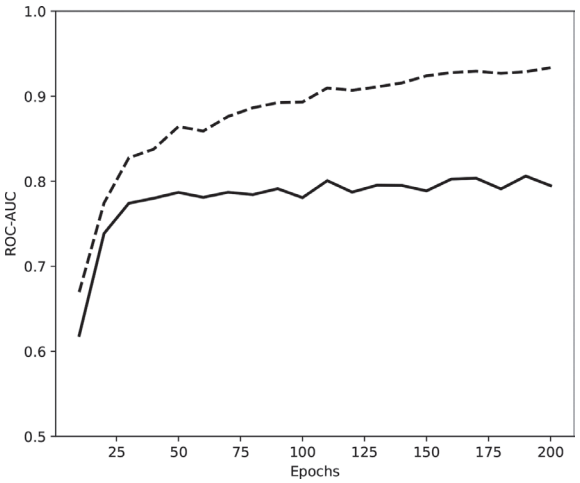


Рис. 6.3 ❖ Эволюция ROC-AUC во время обучения для обучающего набора (пунктир) и проверочного набора (сплошная линия)

РНК-интерференция

В качестве завершающего примера давайте обратимся к РНК. Как и ДНК, это полимер, состоящий из четырех повторяющихся звеньев, называемых основаниями. Фактически три из четырех оснований РНК почти идентичны основаниям ДНК, отличаясь только наличием одного дополнительного атома кислорода. Четвертое основание отличается больше. Вместо тимина (T) у РНК есть основание, которое называется урацил (U). Когда последовательность ДНК транскрибируется в РНК, каждый T заменяется на U.

Основания G и C *комплементарны* друг другу в том смысле, что они имеют выраженную склонность связываться друг с другом. Аналогично, основания A и T (или U) тоже являются комплементарными. Если взять две цепи ДНК или РНК, где каждое основание в одной цепи комплементарно соответствующему основанию в другой цепи, то эти цепи будут склонны слипаться. Этот факт играет ключевую роль во многих биологических процессах, включая транскрипцию, трансляцию и репликацию ДНК при делении клетки.

Комплементарность играет центральную роль в явлении, названном *РНК-интерференцией* (RNA interference). Это явление было обнаружено только в 1990-х годах и получило Нобелевскую премию в 2006 году. Короткий фрагмент РНК, чья последовательность дополняет часть матричной РНК, способен связываться с этой мРНК. Когда это происходит, он «заглушает» мРНК и предотвращает ее трансляцию в белок. Блокирующая трансляцию молекула называется *малой интерферирующей РНК* (миРНК, short interfering RNA).

На самом деле этот процесс устроен еще сложнее. РНК-интерференция является сложным биологическим механизмом, а не просто побочным эффектом двух изолированных цепей РНК, которые могут слипаться. Процесс начинается со связывания миРНК с набором белков, называемым *РНК-индуцированным комплексом молчания* (RNA-induced silencing complex, RISC). RISC использует миРНК в качестве шаблона для поиска и подавления соответствующих мРНК в клетке, что служит как механизмом регуляции активности генов, так и защитой от вирусов.

Это также мощный инструмент для биологии и медицины, позволяющий вам временно «выключить» любой ген, который вы захотите. Вы можете использовать RISC для лечения заболевания или для изучения того, что происходит, когда ген отключен. Просто определите мРНК, которую вы хотите заблокировать, выберите любой ее короткий сегмент и создайте молекулу миРНК с комплементарной последовательностью.

Но все не так просто, как могло показаться. Вы не можете выбрать произвольный сегмент мРНК по своему усмотрению, потому что молекулы РНК – это не просто абстрактные структуры из четырех символов. Это физические объекты с различными свойствами, зависящими от последовательности. Некоторые молекулы РНК более стабильны, чем другие. Иные связывают свои дополнительные последовательности сильнее, чем другие. Некоторые складываются в формы, которые затрудняют их связывание с RISC. Это означает, что некоторые последовательности миРНК работают лучше, чем другие, и если вы хотите использовать РНК-интерференцию в качестве инструмента, вам нужно знать, как выбрать хорошую последовательность.

Биологи разработали множество эвристик для выбора последовательностей миРНК. Они могут сказать, например, что самое первое основание должно быть либо A, либо G и что основания G и C должны составлять от 30 % до 50 % последовательности, и т. д. Эта эвристика полезна, но давайте посмотрим, сможем ли мы добиться большего успеха, используя машинное обучение.

Мы будем обучать нашу модель, используя библиотеку из 2431 молекулы миРНК, каждая длиной в 21 основание[1]. Каждая из молекул была проверена экс-

[1] *Huesken D., Lange J., Mickanin C., Weiler J., Asselbergs F., Warner J., Meloon B., Engel S., Rosenberg A., Cohen D., Labow M., Reinhardt M., Natt F., and Hall J.* Design of a genome-wide siRNA library using an artificial neural network 2005 // **https://doi.org/10.1038/nbt1118**.

периментально и помечена значением в диапазоне от 0 до 1, указывающим на то, насколько эффективно она заставляет молчать свой целевой ген. Малые значения указывают на неэффективные молекулы, в то время как большие значения указывают на более эффективные. Модель принимает последовательность в качестве входных данных и пытается предсказать эффективность.

Код для построения модели выглядит следующим образом:

```
model = dc.models.TensorGraph()
features = layers.Feature(shape=(None, 21, 4))
labels = layers.Label(shape=(None, 1))
prev = features
for i in range(2):
    prev = layers.Conv1D(filters=10, kernel_size=10,
                         activation=tf.nn.relu, padding='same',
                         in_layers=prev)
    prev = layers.Dropout(dropout_prob=0.3, in_layers=prev)
output = layers.Dense(out_channels=1, activation_fn=tf.sigmoid,
                      in_layers=layers.Flatten(prev))
model.add_output(output)
loss = layers.ReduceMean(layers.L2Loss(in_layers=[labels, output]))
model.set_loss(loss)
```

Эта модель очень похожа на рассмотренную ранее модель для предсказывания привязки TF, только с некоторыми отличиями. Поскольку мы работаем с более короткими последовательностями и обучаемся на меньшем количестве данных, мы уменьшили размер модели. Задействовано только два сверточных слоя и 10 фильтров на слой вместо 15. Также нет необходимости в весах, поскольку мы хотим, чтобы каждая выборка вносила одинаковый вклад в процесс оптимизации.

Здесь мы используем другую функцию потерь. Модель связывания TF была классифицирующей моделью. Каждая метка была либо нулем, либо единицей, и мы пытались предсказать одно из двух дискретных значений. Но сейчас у нас регрессионная модель. Метки являются непрерывными числами, и модель пытается вычислить их как можно точнее. Поэтому в качестве функции потерь мы используем евклидово расстояние *L2* и пытаемся минимизировать среднеквадратическую разницу между истинными и предсказанными метками.

Код для обучения модели:

```
train = dc.data.DiskDataset('train_siRNA')
valid = dc.data.DiskDataset('valid_siRNA')
metric = dc.metrics.Metric(dc.metrics.pearsonr, mode='regression')
for i in range(20):
    model.fit(train, nb_epoch=10)
    print(model.evaluate(train, [metric]))
    print(model.evaluate(valid, [metric]))
```

В предыдущем примере мы использовали критерий ROC-AUC в качестве метрики оценки, измеряющей, насколько точно модель может разделить данные на два класса. Эта оценка подходит для проблемы классификации, но не имеет смысла для проблемы регрессии, поэтому в данном случае мы используем коэффициент корреляции Пирсона. Это число от 1 до 1, где 0 означает, что модель вообще

не предоставляет никакой информации, а 1 означает, что она идеально воспроизводит экспериментальные данные.

Результат показан на рис. 6.4. Оценка на проверочном наборе достигает максимального значения 0,65 после 50 эпох обучения. Оценка на обучающем наборе продолжает увеличиваться, но поскольку дальнейшее улучшение оценки на проверочном наборе не происходит, это просто переобучение. Учитывая простоту модели и ограниченный объем обучающих данных, коэффициент корреляции 0,65 можно считать довольно хорошим. Более сложные модели, обученные на больших наборах данных, работают немного лучше, но это уже труднодостижимая точность.

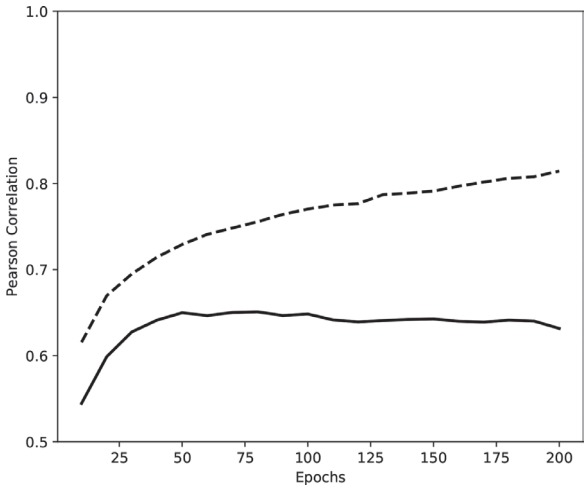


Рис. 6.4 ❖ Эволюция коэффициента корреляции Пирсона во время обучения для тренировочного набора (пунктир) и проверочного набора (сплошная линия)

Заключение

Геном – чрезвычайно сложный механизм с огромным количеством частей, управляющий производством белков и других молекул. Глубокое обучение является мощным инструментом для изучения генома. Нейронные сети могут выявлять тонкие закономерности в геномных данных, формируя представление о функционировании генома, а также служить практическим инструментом для прогнозирования генетических взаимодействий.

Геномика генерирует огромное количество экспериментальных данных, больше чем во многих других областях наук о жизни. Например, одна последовательность генома человека включает более шести миллиардов оснований. Традиционные статистические методы усиленно пытаются найти сигнал, скрытый во всех этих данных. Они часто требуют упрощающих предположений, которые не отражают сложность механизма геномной регуляции. Глубокое обучение уникально подходит для обработки этих данных и углубленного понимания основных функций живых клеток.

Глава 7

Машинное обучение и микроскопия

В этой главе мы познакомим вас с использованием методов глубокого обучения в *микроскопии*. В таких приложениях мы стремимся понять биологическую структуру микроскопического изображения. Например, мы могли бы подсчитать количество клеток определенного типа в данном изображении или попытаться идентифицировать определенные *органеллы*. Микроскопия является одним из самых фундаментальных инструментов в науках о жизни, и достижения в области микроскопии значительно развили понимание устройства человеческого организма. Даже ученые-скептики соглашаются с утверждением «видеть – значит верить», а способность визуально изучать биологические объекты, такие как клетки, способствует интуитивному пониманию основных механизмов жизни. Яркая визуализация клеточных ядер и цитоскелетов (рис. 7.1) формирует гораздо более глубокое понимание, чем сухое обсуждение в учебнике.

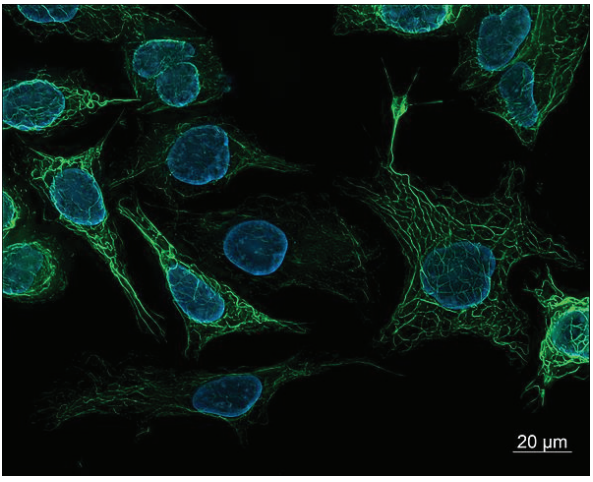


Рис. 7.1 ❖ Человеческие клетки SK8/18-2. Эти клетки окрашивают, чтобы выделить их ядра и цитоскелеты, и наблюдают с использованием флуоресцентной микроскопии. *Источник*: **https://commons.wikimedia.org/wiki/File:SK8-18-2_human_derived_cells,_fluorescence_microscopy_(29942101073).jpg**

Остается открытым вопрос, до какой степени глубокое обучение может изменить микроскопию. До недавнего времени единственным инструментом анализа микроскопических изображений были люди (часто аспиранты или младшие научные сотрудники), вручную проверяющие эти изображения на наличие полезных объектов. Совсем недавно такие инструменты, как CellProfiler, позволили биологам устроить автоматическую конвейерную линию для обработки изображений.

Автоматизированный скоростной микроскопический анализ изображений

Благодаря достижениям в области автоматизации за последние десятилетия в некоторых случаях удается создать системы *автоматизированной скоростной микроскопии*. Эти системы основаны на комбинации простой робототехники (для автоматической обработки препаратов) и алгоритмов обработки изображений. Эти методы обработки изображений позволяют выполнять разделение переднего плана и фона клеток, простой подсчет клеток и другие основные измерения. Кроме того, такие инструменты, как CellProfiler, позволили биологам без опыта программирования создавать новые автоматизированные конвейеры для обработки клеточных данных.

Однако автоматизированные системы микроскопии традиционно сталкиваются с рядом ограничений. Во-первых, существующим системам компьютерного зрения пока не удается решить сложные визуальные задачи, особенно в условиях конвейера. Кроме того, правильная подготовка препаратов для анализа требует значительных усилий со стороны ученого, проводящего эксперимент. По этим причинам автоматическая микроскопия остается относительно нишевой технологией, несмотря на значительный успех в проведении новых сложных экспериментов.

Следовательно, у глубокого обучения есть хорошие шансы для расширения возможностей таких инструментов, как CellProfiler. Если методы глубокого анализа смогут выполнять более сложные исследования, автоматизированная микроскопия станет намного более эффективным инструментом. По этой причине в научном мире наблюдается большой интерес к глубокой микроскопии, что мы и увидим в оставшейся части этой главы.

Надежда на методы глубокого обучения состоит в том, что они позволят системам автоматизированной микроскопии стать значительно более гибкими. Алгоритмы глубокого обучения демонстрируют многообещающую способность выполнять практически любую задачу, которую может выполнять аналитик-человек. Кроме того, ранние исследования показали, что методы глубокого обучения могут значительно расширить возможности дешевого оборудования для микроскопии и позволить недорогим микроскопам проводить анализы, которые в настоящее время возможны только с помощью очень сложных и дорогих аппаратов.

Забегая вперед, скажем, что можно даже обучать глубокие модели, которые имитируют экспериментальные анализы. Такие системы способны прогнозировать результаты экспериментов (в некоторых ограниченных случаях), даже не выполняя рассматриваемый эксперимент! Это очень мощная перспектива, которая вызвала в научном мире большое волнение по поводу потенциала глубоких сетей, основанных на изображениях.

В этой главе мы постараемся научить вас основам глубокой микроскопии. Мы покажем, как системы глубокого обучения могут научиться выполнять простые задачи, такие как подсчет клеток и сегментация клеточных изображений. Кроме того, мы обсудим, как создавать расширяемые системы, которые могли бы обслуживать более сложные конвейеры обработки изображений.

Краткое введение в микроскопию

Прежде чем мы углубимся в алгоритмы, давайте сначала поговорим об основах. *Микроскопия* – это наука об использовании физических систем для просмотра небольших объектов. Традиционно микроскопы представляли собой чисто оптические устройства, использующие тщательно отшлифованные линзы для увеличения изображения *препаратов*. В последнее время для получения изображений с высоким разрешением микроскопия стала все больше опираться на новые технологии, такие как электронные лучи или даже физические зонды.

Микроскопия на протяжении веков тесно связана с науками о жизни. В XVII веке Антони ван Левенгук (Antoni van Leeuwenhoek) использовал собственноручно сконструированные и изготовленные оптические микроскопы для описания микроорганизмов в недоступных ранее малейших деталях (рис. 7.2). Научные достижения того времени во многом зависели от достижений Левенгука в области микроскопии и, в частности, от изобретенной им новой линзы, которая значительно улучшила оптическое разрешение по сравнению с микроскопами, доступными в то время.

Рис. 7.2 ❖ Репродукция микроскопа Левенгука, изготовленная в наше время. Левенгук держал в секрете ключевые детали процесса шлифования линз, и вплоть до 1950-х годов ученые Соединенных Штатов и Советского Союза не могли добиться успешного воспроизведения микроскопа. *Источник*: **https://en.wikipedia.org/wiki/Antonie_van_Leeuwenhoek#/media/File:Leeuwenhoek_Microscope.png**

Изобретение оптических микроскопов высокого разрешения вызвало революцию в микробиологии. Распространение методов микроскопии и возможность видеть клетки, бактерии и другие микроорганизмы способствовали возникновению патогенной модели заболеваний и зарождению микробиологии как самостоятельной науки. Влияние микроскопии на современные науки о жизни трудно переоценить.

Оптические микроскопы бывают либо *простыми* – с единственной линзой, либо *составными*. Составные микроскопы за счет использования системы линз дают большее увеличение, но за это приходится платить усложнением конструкции

и возможными оптическими искажениями. Вплоть до середины XIX века не удавалось создать составные микроскопы, не уступающие по уровню изображения качественным однолинзовым микроскопам! Можно утверждать, что следующий значительный сдвиг в разработке систем оптической микроскопии произошел только в 1980-х годах с появлением цифровых микроскопов, позволяющих записывать изображения, снятые микроскопом, в компьютерное хранилище. Как мы упоминали в предыдущем разделе, автоматизированная микроскопия использует цифровые микроскопы для захвата большого количества изображений. Они могут применяться для проведения крупномасштабных биологических экспериментов с большим количеством экспериментальных воздействий.

Современная оптическая микроскопия

Несмотря на то что оптическая микроскопия существует уже много веков, в этой области все еще случаются значительные инновации. Давайте кратко обсудим некоторые интересные методы в современной оптической микроскопии. Одним из наиболее фундаментальных подходов является метод *оптического секционирования* (optical sectioning). Оптический микроскоп имеет *фокальные плоскости* (focal planes), на одной из которых в определенное время фокусируется микроскоп. Существуют различные методы фокусировки микроскопа на заданной фокальной плоскости, образующей оптический «срез». Эти сфокусированные изображения затем могут быть алгоритмически сшиты для создания изображения с высоким разрешением или даже для трехмерной реконструкции исходного изображения. На рис. 7.3 наглядно показано, как наборы срезов зернышка пыльцы объединяют для получения изображения с высокой точностью.

Популярным методом получения оптических срезов являются *конфокальные микроскопы*. Они используют точечное отверстие, чтобы блокировать свет, поступающий не в фокусе. Преимущество этой техники в том, что она позволяет конфокальному микроскопу достичь лучшей передачи глубины объекта. Смещая фокус микроскопа и выполняя горизонтальное сканирование, можно получить полное изображение всего образца с повышенным оптическим разрешением и контрастностью. Забавный исторический факт: концепция конфокальной визуализации была впервые запатентована пионером искусственного интеллекта Марвином Мински (Marvin Minsky, рис. 7.4).

Качественные микроскопы для оптического секционирования превосходно подходят для получения трехмерных изображений биологических систем, поскольку микроскоп можно поочередно фокусировать на разных фокусных плоскостях и получать послойные сканы поверхности. Затем сфокусированные изображения алгоритмически сшиваются, давая красивые трехмерные реконструкции.

В следующем разделе мы сделаем краткий обзор некоторых фундаментальных ограничений оптической микроскопии и рассмотрим некоторые методы обхода этих ограничений. Этот материал все еще не имеет прямого отношения к глубокому обучению (по причинам, которые мы обсудим), но мы думаем, что он даст вам очень ценное понимание проблем, стоящих сегодня перед микроскопией. Эти знания пригодятся, если вы хотите участвовать в разработке систем микроскопии с машинным обучением следующего поколения. Однако если вам не терпится поработать с программами, мы рекомендуем вам перейти к следующим разделам, где представлены практические примеры кода.

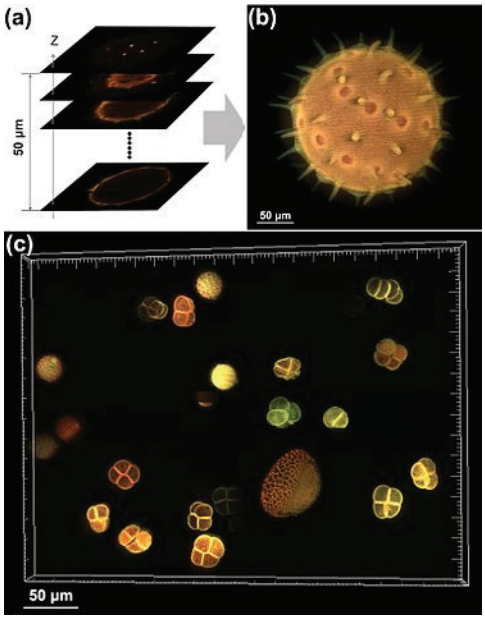


Рис. 7.3 ❖ Получение изображения зернышка пыльцы: (а) флуоресцентные изображения оптических сечений зернышка; (б) комбинированное изображение зернышка; (в) комбинированное изображение группы зернышек пыльцы. *Источник*: **https://commons.wikimedia.org/wiki/File:Optical_sectioning_of_pollen.jpg**

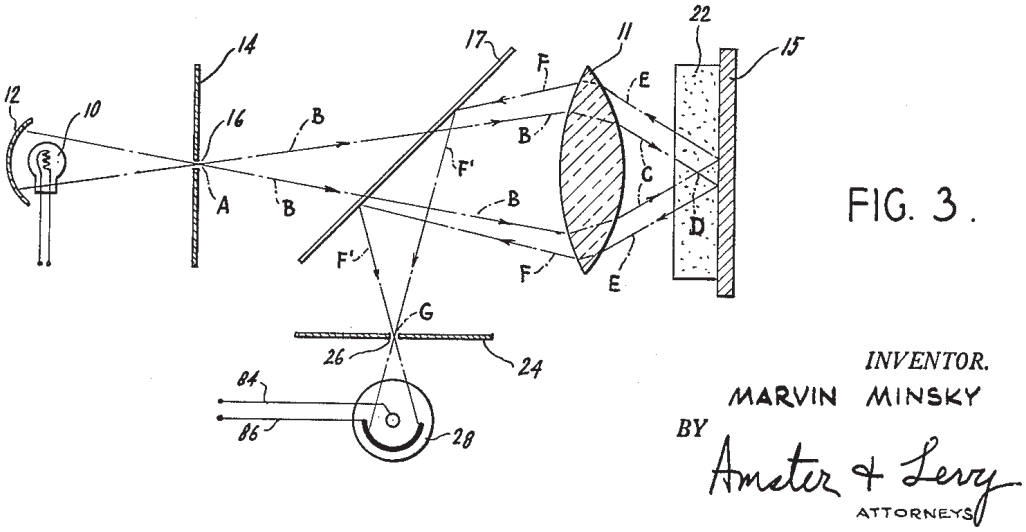


FIG. 3.

Рис. 7.4 ❖ Изображение из оригинального патента Мински, представляющего конфокальный сканирующий микроскоп. Интересно, что Мински больше известен своими работами в области ИИ. *Источник*: **https://en.wikipedia.org/wiki/Confocal_microscopy#/media/File:Minsky_Confocal_Reflection_Microscope.png**

Что не может сделать глубокое обучение в микроскопии?

Интуитивно очевидно, что глубокое обучение полезно для микроскопии, поскольку оно поднимает обработку изображений на новый уровень, а микроскопия – это все о захвате изображений. Но стоит задуматься: каковы пределы возможностей глубокого обучения? В каких областях микроскопии глубокое обучение пока мало на что способно? Как мы увидим позже в этой главе, подготовка образца для микроскопического исследования может потребовать значительных усилий. Кроме того, нужна значительная ловкость рук, потому что экспериментатор должен «подковать блоху» – зафиксировать образец как физический объект. Можем ли мы автоматизировать или ускорить этот процесс с помощью глубокого обучения?

К сожалению, проблема в том, что роботизированные системы все еще очень ограничены. В то время как простые задачи, такие как конфокальное сканирование препарата, легко поддаются автоматизации, очистка и подготовка препарата требуют значительного человеческого опыта. Маловероятно, что любые роботизированные системы в ближайшем будущем смогут выполнять такую работу.

Всякий раз, когда вы слышите прогнозы о блестящем будущем глубокого обучения, очень полезно держать в уме такие примеры, как подготовка препаратов. Многие болевые точки в науках о жизни связаны с трудоемкой ручной работой, такой как подготовка препаратов, непосильной для современного машинного обучения. И вряд ли что-то изменится в ближайшие несколько лет.

Дифракционный предел

При изучении нового физического инструмента, такого как микроскоп, будет очень полезно начать с попытки понять пределы его возможностей. Чем ограничены возможности микроскопов? Этот вопрос был глубоко изучен предыдущими поколениями физиков, но есть и новые сюрпризы. Первое, с чего нужно начать, – это *дифракционный предел*, теоретический предел разрешения, достижимого с помощью микроскопа.

$$d = \frac{\lambda}{2n\sin\theta}.$$

Величина $n\sin\theta$ часто обозначается как *числовая апертура* (numerical aperture, NA). Здесь λ – длина волны светового излучения. В этой формуле есть неявные предположения. Мы предполагаем, что препарат освещается неким видом света. Давайте кратко рассмотрим спектр световых волн (рис. 7.5).

Заметим, что видимый свет составляет только крошечную часть этого спектра. В принципе, казалось бы, если использовать световое излучение с достаточно малой длиной волны, то можно значительно улучшить оптическое разрешение. В некоторой степени так и есть. Был разработан ряд микроскопов, в которых используются короткие электромагнитные волны более высокой энергии. Например, достоинство ультрафиолетового микроскопа в том, что ультрафиолетовые лучи имеют меньшую длину волны и обеспечивают более высокое разрешение. Разве мы не можем пойти дальше и использовать излучение с еще меньшей длиной волны? Например, почему бы не сделать рентгеновский или гамма-микроскоп? Основной проблемой здесь является *фототоксичность* (phototoxicity). Свет с малой длиной волны несет высокую энергию, способную разрушить структуру биологического объекта. Кроме того, коротковолновые излучения опасны для экспериментатора и требуют применения специальных защитных средств.

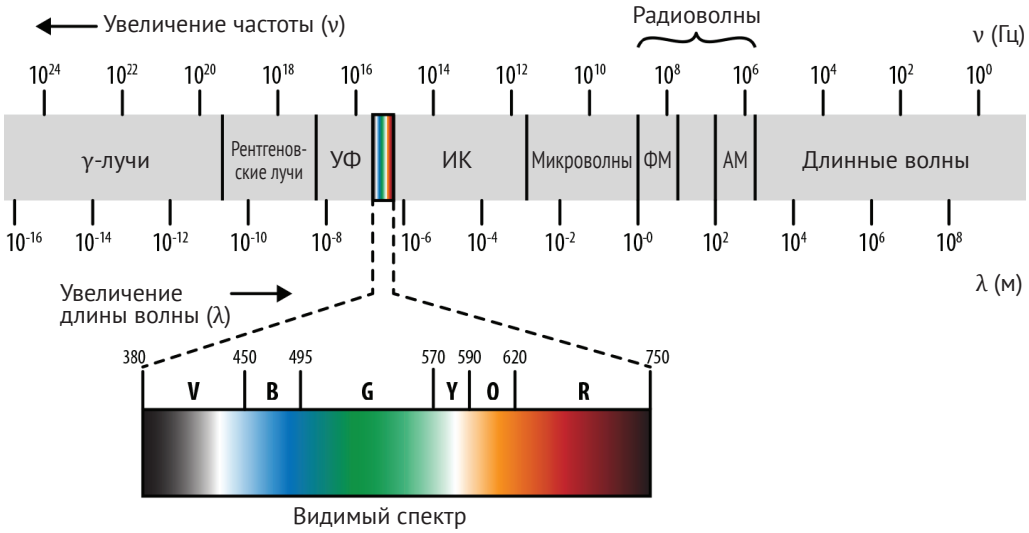


Рис. 7.5 ❖ Длина волны света. Обратите внимание, что источники света с малой длиной волны, такие как рентгеновские лучи, становятся все более энергоемкими. В результате они часто уничтожают деликатные биологические объекты

К счастью, существуют и другие методы обхода дифракционного предела. В одном методе используют электроны, в определенном контексте тоже имеющие волновую природу. В другом методе применяют физические зонды вместо света. Еще один способ заключается в использовании электромагнитных волн *ближнего поля* (near field). И наконец, для снижения дифракционного предела можно использовать трюки с подсветкой нескольких *флуорофоров* (fluorophores). Мы обсудим эти методы в следующих разделах.

Электронная и атомно-силовая микроскопия

В 1930-х годах появление электронного микроскопа вызвало резкий скачок в развитии современной микроскопии. В электронном микроскопе вместо видимого света работает поток электронов. Поскольку длины волн электронов намного меньше, чем у видимого света, использование электронных лучей вместо световых волн позволяет получить существенно более детальные изображения. Как такое возможно? Разве электроны не являются частицами? Помните, что материя может проявлять волновую природу. Это *длина волны де Бройля*, впервые предложенная Луи де Бройлем (Louis de Broglie).

$$\lambda = \frac{h}{p} = \frac{h}{mv}.$$

Здесь h – постоянная Планка; m и v – масса и скорость рассматриваемой частицы. (Физики могут заметить, что приведенная выше формула не учитывает релятивистские эффекты. Существуют модифицированные версии формулы, учитывающие эти эффекты.) Электронные микроскопы используют волнообразную природу электронов для изображения физических объектов. Длина волны

электрона зависит от его энергии, но для получения субнанометровых волн достаточно обычной электронной пушки. Вернувшись к формуле дифракционного предела, легко убедиться, что электронная микроскопия может быть мощным инструментом. Первые прототипы электронных микроскопов были сконструированы в начале 1930-х годов. Хотя конструкция электронного микроскопа значительно усовершенствована, современные электронные микроскопы основаны на прежних принципах (рис. 7.6).

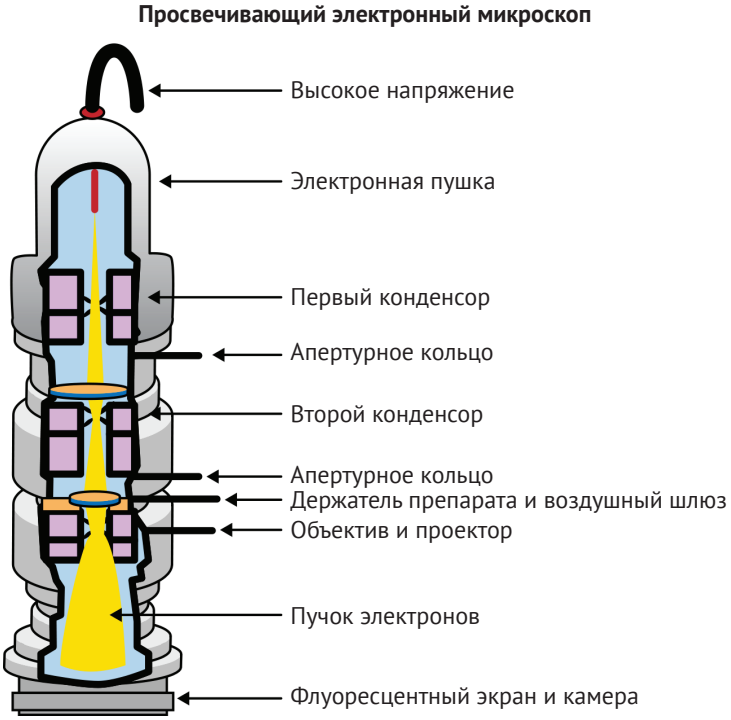


Рис. 7.6 ❖ Схема компонентов современного *просвечивающего электронного микроскопа* (ПЭМ, transmission electron microscope, TEM).
Источник: **https://commons.wikimedia.org/wiki/File:Electron_Microscope.png**

К сожалению, мы не полностью обошли здесь проблему фототоксичности. То есть чтобы получить электроны с очень маленькой длиной волны, нам нужно увеличить их энергию. Но при высокой энергии мы снова уничтожаем препараты. Кроме того, процесс подготовки образцов для электронного микроскопа сам по себе бывает довольно сложным. Тем не менее электронные микроскопы позволяют получить потрясающие изображения микроскопических систем (рис. 7.7). Сканирующие электронные микроскопы, расширяющие поле зрения за счет сканирования объекта, позволяют получать изображения с разрешением всего 1 нанометр.

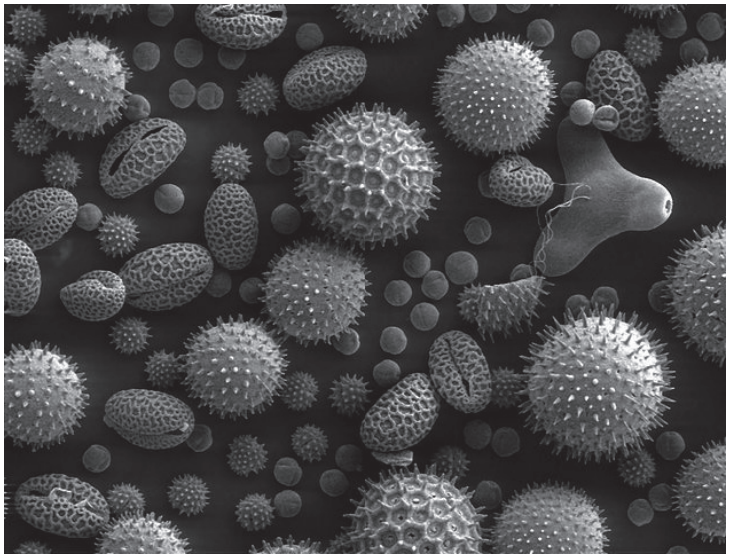


Рис. 7.7 ❖ Пыльца увеличена в 500 раз
с помощью сканирующего электронного микроскопа.
Источник: **https://commons.wikimedia.org/wiki/File:Misc_pollen.jpg**

Атомно-силовая микроскопия (АСМ, atomic force microscopy, AFM) – это еще один способ преодоления оптического дифракционного предела. В этом методе применяется так называемый *кантилевер* (cantilever), физически исследующий поверхность препарата. Прямой физический контакт между кантилевером и объектом исследования позволяет получать изображения с разрешением в долях нанометра. Иногда удается получить изображение отдельных атомов! Атомно-силовые микроскопы также обеспечивают трехмерные изображения поверхности благодаря прямому контакту кантилевера с поверхностью.

Вообще, силовая микроскопия – это метод, изобретенный относительно недавно. Первые атомно-силовые микроскопы были изобретены только в 1980-х годах, поскольку лишь современные нанотехнологии позволяют изготовить зонды-кантилеверы. Пока еще рано говорить о применении АСМ в науках о жизни. Уже проведены некоторые исследования по визуализации клеток и биомолекул с помощью зондов АСМ, но в целом технология остается на раннем этапе развития.

Микроскопия сверхвысокого разрешения

В данной главе мы обсудили ряд методов увеличения дифракционного предела. В этих методах применяется свет с меньшей длиной волны и специальные физические датчики. Однако во второй половине XX века настоящим научным прорывом стало изобретение целого семейства методов для преодоления дифракционного предела. Эти методы в совокупности называются *микроскопией сверхвысокого разрешения*. Мы кратко рассмотрим некоторые из них.

Функциональная микроскопия сверхвысокого разрешения использует физические свойства светоизлучающих веществ, встроенных в препарат. Например, в биологической микроскопии часто используют флуоресцентные метки (подроб-

нее об этом позже), чтобы выделить конкретные биологические молекулы. Функциональные методы сверхразрешения позволяют обнаруживать эти излучатели света стандартными оптическими микроскопами. В свою очередь, функциональные методы можно разделить на семейства *детерминированных* и *стохастических* методов.

Детерминированное сверхразрешение основано на том, что некоторые светоизлучающие вещества дают нелинейный отклик на возбуждение. Что это означает на практике? Идея состоит в том, что произвольная фокусировка на конкретном излучателе света может быть достигнута путем «выключения» других излучателей, находящихся поблизости. За этим кроется достаточно запутанная физика, но такие методы, как микроскопия с *подавлением спонтанного излучения* (stimulated emission depletion, STED), успешно применяются на практике.

Стохастическая микроскопия сверхвысокого разрешения основана на хаотичном перемещении светоизлучающих молекул в биологических системах. Это означает, что если движение светоизлучающей частицы наблюдать достаточно долго, то за счет усреднения измерений можно дать достаточно точную оценку ее истинного положения. Существует ряд методов (таких как STORM, PALM, BALM), развивающих эту основную идею. Микроскопия сверхвысокого разрешения оказала большое влияние на современную биологию и химию, потому что она позволяет при помощи относительно дешевого оптического оборудования исследовать поведение наноразмерных систем. Нобелевскую премию 2014 года по химии получили пионеры функциональных методов сверхразрешения.

Глубокие методы сверхвысокого разрешения

В некоторых недавних публикациях начали говорить о применении методов глубокого обучения для воссоздания изображений с суперразрешением[1]. Эти методы обещают увеличение скорости микроскопии сверхвысокого разрешения на несколько порядков за счет восстановления редких, быстро полученных изображений. Эти методы пока остаются на раннем этапе разработки, но тем не менее выглядят перспективной областью применения для глубокого обучения.

Микроскопия ближнего поля (near field microscopy) является еще одним методом сверхразрешения, который использует локальное электромагнитное излучение препарата. Эти *нераспространяющиеся волны* (evanescent waves) не подчиняются дифракционному пределу, поэтому возможно более высокое разрешение. Однако проблема заключается в том, что микроскоп должен регистрировать излучение очень близко к препарату, в пределах одной длины волны света. Это означает, что методы ближнего поля плохо применимы на практике. Совсем недавно появилась возможность создавать *метаматериалы* (metamaterials), которые имеют отрицательный показатель преломления. На практике это означает, что ближние поля можно усилить, чтобы исследовать их на большем расстоянии от препарата. Исследования в этой области еще только начинаются, но первые результаты выглядят впечатляюще.

---

[1] *Ouyang Wei, et al*. Deep Learning Massively Accelerates Super-Resolution Localization Microscopy // Nature Biotechnology 36 (April 2018): 460–468. URL: **https://doi.org/10.1038/nbt.4106**.

Глубокое обучение и дифракционный предел

В научных публикациях появились интригующие намеки на то, что глубокое обучение может способствовать развитию микроскопии сверхвысокого разрешения. Несколько ранних работ показало, что алгоритмы глубокого обучения способны ускорить построение изображений с суперразрешением или обеспечить эффективное суперразрешение с помощью относительно дешевого оборудования. Мы упомянули одну такую статью в предыдущем разделе.

Эти идеи звучат весьма убедительно, поскольку глубокое обучение успешно использовалось для решения похожих задач, таких как удаление размытия изображения[1]. Это означает, что есть надежда создать надежный набор инструментов сверхразрешения, основанных на глубоком обучении. К сожалению, в настоящее время исследования только начинаются, и пригодные к применению инструменты еще не созданы. Однако мы надеемся, что это положение изменится в ближайшие годы.

Подготовка биологических препаратов для микроскопии

Одним из критически важных шагов в применении микроскопии в науках о жизни является подготовка препаратов для микроскопии (так называемая *пробоподготовка*). Зачастую это весьма нетривиальный процесс, требующий сложного экспериментального оборудования. В этом разделе мы обсудим ряд методов подготовки препаратов, а также отметим, что может пойти не так и создать неожиданные экспериментальные артефакты.

Окрашивание

Самые ранние оптические микроскопы показывали просто увеличенные изображения микроскопических объектов. Благодаря этому мы стали намного лучше понимать внутреннее устройство объектов, но существенное ограничение оптической микроскопии заключалось в невозможности выделить отдельные области изображения. Это привело к появлению технологии *избирательного химического окрашивания* (chemical staining), позволяющей выделять контрастные области изображения, так называемые «окрашенные пятна».

Существует большое разнообразие стандартных красителей, которые предназначены для обработки различных типов препаратов. Сами процедуры окрашивания могут быть довольно сложными и включать несколько этапов. Окраска отдельных областей имеет большое значение с научной точки зрения. Например, в зависимости от реакции на хорошо известную *окраску по Граму* (Gram stain) бактерии обычно классифицируют как *грамположительные* или *грамотрицательные*. Сегментирование и маркировка грамположительных и грамотрицательных бактерий в биологических образцах может быть интересной задачей для системы глубокого обучения. Почему это необходимо? Если у нас в разработке есть потенциальный антибиотик, обычно требуется по отдельности изучить его влияние на грамположительные и грамотрицательные бактерии.

---

[1] *Tao Xin, et al*. Scale-Recurrent Network for Deep Image Deblurring. 2018 // **https://arxiv.org/pdf/1802.01770.pdf**.

Почему пробоподготовка должна беспокоить разработчиков?

Некоторые из читателей этого раздела могут оказаться инженерами, заинтересованными в разработке и развертывании систем автоматизированной глубокой микроскопии. Они наверняка зададут обоснованный вопрос: зачем нам беспокоиться о подготовке биологических препаратов?

Если вы действительно сосредоточены на разработке систем автоматизированной глубокой микроскопии, вероятно, вам больше всего помогут практические примеры кода в конце этой главы. Тем не менее понимание основ пробоподготовки может в дальнейшем избавить вас от головной боли и даст вам словарный запас для эффективного общения с коллегами-биологами. Если биологи потребуют, чтобы вы добавили поля метаданных для «пятен», вы не будете озадачены этим запросом и догадаетесь, о чем они на самом деле просят. Это стоит нескольких страниц чтения!

Разработка антибактериальных агентов для грамотрицательных бактерий

Разработка эффективных антибиотиков для грамотрицательных бактерий сегодня относится к одной из основных проблем фармацевтической промышленности. Грамотрицательные бактерии имеют дополнительную клеточную стенку, которая препятствует проникновению обычных антибактериальных агентов, нацеленных на клеточные стенки пептидогликана грамположительных бактерий.

Эта проблема со временем становится более актуальной, поскольку многие штаммы бактерий приобретают *грамотрицательную резистентность* за счет горизонтального переноса генов, и после десятилетий успешного лечения вновь возрастает смертность от бактериальных инфекций.

Для решения этой проблемы можно попробовать объединить уже знакомые вам методы глубокого обучения, предназначенные для целевого «конструирования» молекул, и методы визуализации, обсуждаемые в этой главе. Мы призываем заинтересованных разработчиков глубже изучить эту тему.

Фиксация препаратов

Большие биологические препараты, такие как ткани организма, обычно быстро разлагаются, если оставить их «как есть». Метаболические процессы, протекающие в препарате, будут поглощать и повреждать структуру органов, клеток и органелл. Процесс *фиксации* призван остановить разложение и стабилизировать содержимое препарата, чтобы можно было получить достоверное изображение. С этой целью был разработан ряд *фиксирующих агентов*, или *фиксаторов*. Одна из основных функций фиксаторов – денатурировать белки и отключить протеолитические ферменты. Если такие ферменты не заблокировать, они будут «поедать» препарат.

Кроме того, фиксация должна уничтожить микроорганизмы, способные повредить образец. Например, при *тепловой фиксации* препарат пропускается через бунзеновскую горелку, хотя такой процесс может повредить внутреннюю структуру препарата. Другая распространенная техника – *фиксация погружением*. Препараты обычно погружают в фиксирующий раствор и оставляют для пропитывания. Например, препарат может быть подвергнут пропитыванию холодным формалином в течение 24 часов.

Перфузия – это метод фиксации препаратов тканей относительно крупных животных, таких как мыши. Экспериментаторы вводят фиксатор в сердце и ждут, когда мышь умрет, прежде чем извлекать пробу ткани. Этот процесс позволяет фиксирующему агенту естественным образом распространиться по организму и часто дает превосходные результаты.

Секционирование препаратов

Важным этапом изучения биологического препарата является получение тонкого среза для микроскопии. Существует целый ряд оригинальных инструментов, облегчающих этот процесс, включая *микротом* (рис. 7.8), нарезающий биологические препараты на тонкие пластинки. Микротом имеет свои ограничения; таким способом трудно нарезать очень маленькие предметы. Для небольших объектов имеет смысл использовать другие методы, например конфокальную микроскопию.

Стоит на минутку остановиться и пояснить, почему даже специалистам по информатике полезно знать, что существуют такие устройства, как микротом. Допустим, вы инженер и строите конвейерную систему для обработки серии изображений препарата мозга. Вероятно, препарат мозга был нарезан на тонкие пластинки с помощью микротома или подобного режущего устройства. Знание физической природы этого процесса поможет вам разработать схему для последовательной организации изображений и даже восстановления объемной структуры объекта.

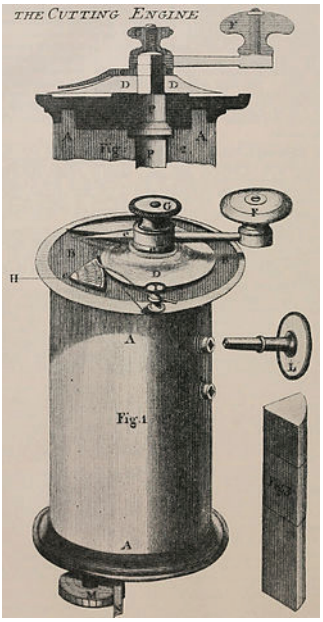


Рис. 7.8 ❖ Ранняя схема 1770 года, изображающая микротом.
Источник: **https://commons.wikimedia.org/wiki/File:Cummings_1774_Microtome.jpg**

Флуоресцентная микроскопия

Флуоресцентный микроскоп – это оптический микроскоп, использующий явление *флуоресценции*. Это физическое явление, при котором препарат поглощает свет на одной длине волны и излучает его на другой длине волны. Например, некоторые минералы флуоресцируют при воздействии ультрафиолетового света. Это явление особенно интересно проявляет себя в области биологии. Многие бактерии естественным образом флуоресцируют, когда их белки поглощают высокочастотный свет высокой энергии и излучают низкочастотный свет.

Флуорофоры и флуоресцентные метки

Флуорофор – это химическое соединение, способное излучать свет с определенной длиной волны. Такие флуорофоры чрезвычайно важны и популярны в биологии, поскольку они позволяют экспериментаторам выделять конкретные компоненты данной клетки. В экспериментах флуорофор обычно наносят в виде метки-красителя на конкретную клетку. Молекулярная структура распространенного флуорофора показана на рис. 7.9.

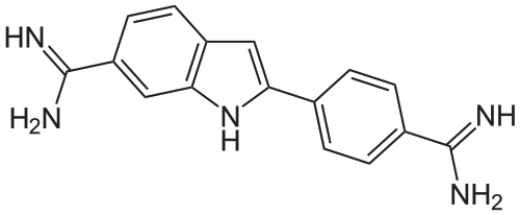


Рис. 7.9 ❖ DAPI (4',6-диамидино-2-фенилиндол) является обычным флуоресцентным красителем, который связывается с богатыми аденинтимином участками ДНК. Поскольку он проникает через клеточные мембраны, то обычно применяется для окрашивания внутренностей клеток. *Источник*: **https://commons.wikimedia.org/wiki/File:DAPI.svg**

Флуоресцентная маркировка (fluorescent tagging) – это метод прикрепления флуорофора к биомолекуле, представляющей интерес в организме. Почему это полезно? В микроскопии обычно требуется выделить определенную часть изображения. Флуоресцентные метки очень эффективно решают эту задачу.

Флуоресцентная микроскопия оказалась огромным благом для биологических исследований, поскольку она позволяет исследователям находить и увеличивать конкретные фрагменты в данном биологическом препарате, а не иметь дело со всей пробой. Использование маркировки зачастую оказывается неоценимым при изучении отдельных клеток или отдельных молекул внутри клетки. На рис. 7.10 показано, как флуоресцентное окрашивание применяется для избирательной визуализации определенных хромосом в ядре клетки человека.

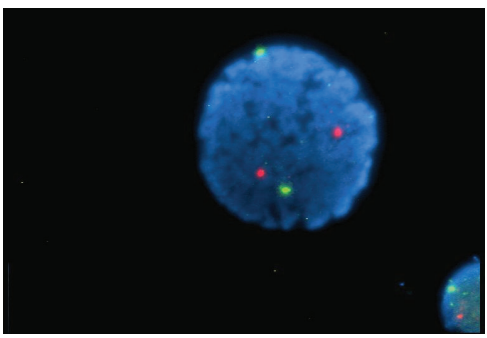


Рис. 7.10 ❖ Изображение ядра лимфоцита человека с хромосомами 13 и 21, окрашенными DAPI (популярный флуоресцентный краситель). *Источник*: **https://commons.wikimedia.org/wiki/File:FISH_13_21.jpg**

Флуоресцентная микроскопия может быть очень точным инструментом для отслеживания отдельных событий связывания молекул. Например, если вспомнить посвященную биофизике главу 5, события связывания белков с лигандами можно обнаружить с помощью флуоресцентного анализа.

Артефакты пробоподготовки

Важно отметить, что пробоподготовка бывает очень сложным процессом. Обычно при подготовке исходного препарата возникают искажения в объекте исследования, что может привести к некоторой путанице. В качестве любопытного примера рассмотрим случай обнаружения мезосомы, обсуждаемый в предупреждении ниже.

Мезосома – воображаемая органелла

Процесс фиксации клетки для электронной микроскопии привел к появлению важнейшего артефакта – *мезосомы* у грамположительных бактерий. Процесс подготовки препарата для электронного микроскопа вызвал разрушение клеточной стенки, которое долго считалось естественной структурой, а не артефактом.

Имейте в виду, что аналогичные артефакты могут существовать в ваших собственных препаратах. Кроме того, вполне вероятно, что ваша модель обучится на артефактах вместо обнаружения реальных биологических объектов.

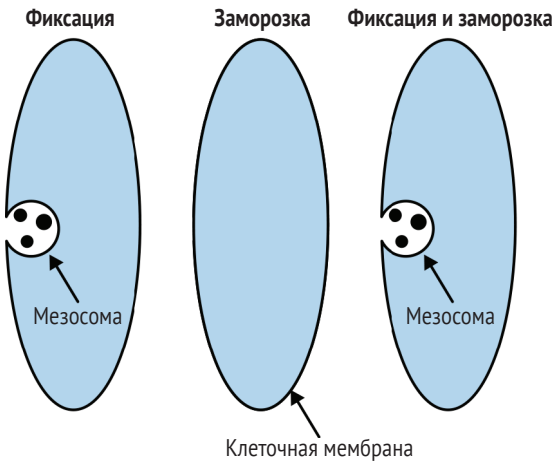


Рис. 7.11 ❖ Мезосомы оказались артефактами, возникшими при подготовке препарата для электронной микроскопии, но когда-то считались реальными структурами в клетках. *Источник*: **https://en.wikipedia.org/wiki/Mesosome#/media/File:Mesosome_formation.svg**

Отслеживание происхождения микроскопических препаратов

Когда вы разрабатываете системы для обработки данных микроскопии, важно отслеживать происхождение препаратов. Каждое изображение должно быть снабжено информацией об условиях, в которых оно было получено. Информация может включать название физического устройства для захвата изображения, фамилию техника, который управлял устройством, препарат, который был обработан, и территориальное происхождение препарата. Биология чрезвычайно сложна для «отладки». Такие проблемы, как мезосома, могут не отслеживаться даже десятилетиями. По этой причине сохранение адекватных метаданных о происхождении изображений может спасти вас и вашу команду от серьезных проблем в будущем.

Применение глубокого обучения в микроскопии

В этом разделе мы кратко рассмотрим различные применения глубокого обучения в микроскопии. Мы обсудим некоторые примеры применений, такие как подсчет клеток, сегментация клеток и построение вычислительных экспериментов. Как мы уже отмечали ранее в этой главе, это весьма ограниченный перечень применений. Тем не менее он даст вам понимание, необходимое при разработке новых собственных приложений для глубокой микроскопии.

Подсчет клеток

Вот вам простая задача – подсчитать количество клеток, которые видны на данном изображении. Вы можете обоснованно спросить, что же интересного в этой задаче? Для ряда биологических экспериментов бывает весьма полезно отслеживать количество клеток, выживших после определенного вмешательства. Допустим, клетки взяты из линии раковых клеток, и вмешательством является применение противоракового вещества. Успешное вмешательство уменьшит количество живых раковых клеток, поэтому было бы полезно иметь точную систему глубокого обучения, способную подсчитывать количество живых клеток без участия человека.

Что такое клеточная линия?

В биологии часто изучают клетки определенного типа. Разумеется, чтобы провести эксперимент на клетках, нужно получить достаточное количество таких клеток, то есть *клеточную линию*. Клеточные линии представляют собой клетки, культивируемые из заданного источника и способные стабильно расти в лабораторных условиях.

Клеточные линии использовались в бесчисленных биологических исследованиях, но часто есть серьезные сомнения в достоверности результатов, получаемых на таких клеточных линиях. Прежде всего удаление клетки из ее естественной среды зачастую радикально меняет ее биологию. Обнаруживается все больше доказательств того, что окружение клетки фундаментально определяет ее реакцию на внешние раздражители.

Что еще хуже, клеточные линии бывают серьезно загрязнены. Клетки одной клеточной линии часто загрязняют клетки другой клеточной линии, поэтому результаты по клеточной линии «рак молочной железы» могут фактически ничего не сказать исследователю о раке молочной железы!

По этим причинам исследования на клеточных линиях часто рассматриваются с осторожностью. Результаты экспериментов на таких клеточных линиях часто являются лишь стимулом для последующего дублирования результатов с помощью испытаний на животных или людях. Тем не менее исследования клеточных линий обеспечивают бесценный легкий старт в биологических исследованиях и применяются повсеместно.

Как показано на рис. 7.12, микроскопическое изображение может значительно отличаться от стандартного фотографического изображения, поэтому не сразу очевидно, что такие технологии, как сверточные нейронные сети, могут быть адаптированы к таким задачам, как подсчет клеток. К счастью, многочисленные экспериментальные работы показали, что сверточные сети хорошо учатся на наборах данных микроскопии.

Рис. 7.12 ❖ Препараты клеток дрозофилы. Обратите внимание, что внешний вид препарата на микроскопических изображениях может значительно отличаться от фотографического изображения. *Источник*: Cell Image Library (**http://cellimagelibrary.org/images/21780**)

Внедрение подсчета клеток в DeepChem

В этом разделе мы рассмотрим создание модели глубокого обучения подсчету клеток с использованием DeepChem. Давайте начнем с загрузки и настройки набора данных для подсчета клеток. Мы воспользуемся набором Broad Bioimage Benchmark Collection (BBBC) – готовым набором полезных данных клеточной микроскопии.

Наборы данных BBBC

Наборы данных BBBC содержат полезную коллекцию аннотированных биологических изображений из различных клеточных анализов. Если вы работаете над обучением своих моделей глубокой микроскопии, эта коллекция может оказаться весьма полезным ресурсом. DeepChem имеет набор инструментов для обработки изображений, чтобы упростить работу с этим и подобными наборами данных. В частности, класс ImageLoader облегчает загрузку наборов данных микроскопии.

Обработка наборов данных изображений

Изображения обычно хранятся на диске в стандартных форматах файлов изображений, таких как PNG, JPEG и т. д. Конвейер обработки наборов данных обычно считывает эти файлы с диска и преобразует их в подходящее представление в памяти – как правило, это многомерный массив. В конвейерах обработки Python этот массив часто является просто массивом NumPy. Для изображения высотой N пикселей, шириной M пикселей и с тремя цветными каналами RGB вы получите массив формы (N, M, 3). Если у вас есть 10 таких изображений, эти изображения обычно объединяются в массив формы (10, N, M, 3).

Прежде чем мы начнем загружать набор данных в DeepChem, нам сначала нужно будет скачать его на локальную машину. Наборы данных достаточно большие, почти 2 ГБ, поэтому убедитесь, что на машине для разработки достаточно свободного места.

```
wget https://data.broadinstitute.org/bbbc/BBBC005/BBBC005_v1_images.zip
unzip BBBC005_v1_images.zip
```

Скачав набор данных на локальный компьютер, теперь вы можете загрузить этот набор в DeepChem с помощью `ImageLoader`:

```
image_dir = 'BBBC005_v1_images'
files = []
```

```
labels = []
for f in os.listdir(image_dir):
 if f.endswith('.TIF'):
  files.append(os.path.join(image_dir, f))
  labels.append(int(re.findall('_C(.*?)_', f)[0]))
loader = dc.data.ImageLoader()
dataset = loader.featurize(files, np.array(labels))
```

Этот код проходит по загруженному каталогу и извлекает файлы изображений. Метки закодированы в именах файлов, поэтому мы используем простое регулярное выражение для извлечения количества клеток в каждом изображении. Мы используем `ImageLoader` для преобразования содержимого каталога в набор данных DeepChem.

Теперь разделим этот набор данных на обучающий, проверочный и оценочный наборы:

```
splitter = dc.splits.RandomSplitter()
train_dataset, valid_dataset, test_dataset = splitter.train_valid_test_split(dataset, seed=123)
```

Данные готовы, и мы можем определить саму модель. В данном случае давайте создадим простую сверточную архитектуру с полностью подключенным слоем в конце:

```
learning_rate = dc.models.tensorgraph.optimizers.ExponentialDecay(0.001, 0.9, 250)
model = dc.models.TensorGraph(learning_rate=learning_rate, model_dir='model')
features = layers.Feature(shape=(None, 520, 696))
labels = layers.Label(shape=(None,))
prev_layer = features
for num_outputs in [16, 32, 64, 128, 256]:
  prev_layer = layers.Conv2D(num_outputs, kernel_size=5, stride=2, in_layers=prev_layer)
output = layers.Dense(1, in_layers=layers.Flatten(prev_layer))
model.add_output(output)
loss = layers.ReduceSum(layers.L2Loss(in_layers=(output, labels)))
model.set_loss(loss)
```

Обратите внимание, что мы используем функцию L2Loss для обучения модели на регрессионной задаче. Несмотря на то что количество клеток является целым числом, у нас ведь нет естественного верхнего предела количества клеток в изображении.

Обучение этой модели потребует некоторых вычислительных усилий (подробнее об этом позже). Поэтому для базовых экспериментов мы рекомендуем воспользоваться нашей предварительно обученной моделью. Эта модель уже была обучена на наборе данных и может использоваться, чтобы делать прогнозы «из коробки». В файловом архиве книги есть инструкции по загрузке предварительно обученной модели. После скачивания предварительно обученные веса можно загрузить в модель:

```
model.restore()
```

Давайте окунем эту предварительно обученную модель в работу. Первым делом вычислим среднюю ошибку прогнозирования в нашем тестовом наборе для задачи подсчета ячеек:

```
y_pred = model.predict(test_dataset).flatten()
print(np.sqrt(np.mean((y_pred-test_dataset.y)**2)))
```

Какую точность вы получаете, когда пытаетесь запустить модель?

Ну хорошо, а как обучить эту модель для себя? Вы можете сформировать модель, пройдя 50 эпох обучения на наборе данных. Это достаточно серьезная вычислительная задача. На хорошем графическом процессоре она завершается в течение часа или около того. Вряд ли стоит начинать обучение на обычном процессоре.

```
model.fit(train_dataset, nb_epoch=50)
```

Мы рекомендуем проверить точность этой обученной модели на проверочном и оценочном наборах. Ваша точность соответствует точности предварительно подготовленной модели?

Клеточная сегментация

Задача *клеточной сегментации* (cell segmentation) состоит в том, чтобы аннотировать изображение клеточной микроскопии, отмечая, где видны клетки и где присутствует только фон. Почему это полезно? Если вы вспомните грамположительные и грамотрицательные бактерии, вы, вероятно, сможете догадаться, почему востребована автоматизированная система для разделения двух типов бактерий. Оказывается, подобные проблемы присущи всей клеточной микроскопии в целом. (Подобные проблемы возникают и в других областях визуализации, как мы увидим в главе 8.)

Маски сегментации обеспечивают значительно более точное разрешение и позволяют проводить более точный анализ, чем подсчет клеток. Например, было бы полезно понять, какая часть данной предметной пластины покрыта клетками. Такой анализ легко выполнить после создания масок сегментации. На рис. 7.13 приведен пример маски сегментации, созданной из синтетического набора данных.

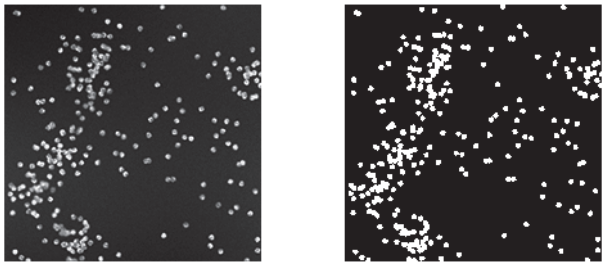


Рис. 7.13 ❖ Синтетический набор клеточных данных (слева) и маски переднего плана и фона, обозначающие места появления клеток на изображении. *Источник*: Broad Institute (**https://data.broadinstitute.org/bbbc/BBBC005/**)

При этом сегментация требует от модели машинного обучения значительно большего, чем банальный подсчет клеток. Способность точно различать клеточные и неклеточные области требует большей точности в обучении. Поэтому неудивительно, что применять машинное обучение в клеточной сегментации

по-прежнему труднее, чем в подсчете клеток. Мы будем экспериментировать с моделью сегментации позже в этой главе.

Откуда берутся обучающие маски сегментации?

Стоит отметить, что маски сегментации являются сложными объектами. В целом для генерации таких масок не существует хороших алгоритмов, кроме методов глубокого обучения. Как тогда мы можем получить обучающие данные, необходимые для совершенствования техники глубокой сегментации? Одним из вариантов является использование синтетических данных, как на рис. 7.13. Поскольку клеточное изображение генерируется синтетически, маска также может генерироваться синтетическим путем. Это полезный трюк, но имеющий очевидные ограничения, поскольку он ограничивает наши изученные методы сегментации подобными изображениями.

Более общее решение заключается в генерации аннотированных масок сегментации при помощи человека. Подобные процедуры широко используются для обучения беспилотных автомобилей. В этой задаче очень важно найти сегментацию, в которой отмечены пешеходы и дорожные знаки, а миллионы людей используются для получения необходимых обучающих данных (знаменитые картинки Google CAPTCHA). Поскольку машинная микроскопия приобретает все большее значение, вполне вероятно, что подобная потребность в человеческом участии станет критически узким местом.

Реализация клеточной сегментации в DeepChem

В этом разделе мы обучим модель клеточной сегментации на том же наборе данных BBBC005, который использовали ранее для задачи подсчета клеток. Здесь есть решающая тонкость. В задаче подсчета клеток меткой каждого обучающего изображения является просто количество клеток. Однако в задаче клеточной сегментации каждая метка сама является изображением! Это означает, что модель клеточной сегментации на самом деле является формой *преобразователя изображения*, а не простой моделью классификации или регрессии. Давайте начнем с получения обучающего набора данных. Нам нужно будет получить маски сегментации с веб-сайта BBBC:

```
wget https://data.broadinstitute.org/bbbc/BBBC005/BBBC005_v1_ground_truth.zip
unzip BBBC005_v1_ground_truth.zip
```

Эталонные экспериментальные данные (ground-truth data) имеют размер около 10 МБ, поэтому их легче скачать, чем полный набор данных BBBC005. Теперь давайте загрузим этот набор данных в DeepChem. К счастью для нас, ImageLoader настроен для обработки наборов данных сегментации изображений без особых хлопот:

```
image_dir = 'BBBC005_v1_images'
label_dir = 'BBBC005_v1_ground_truth'
rows = ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P')
blurs = (1, 4, 7, 10, 14, 17, 20, 23, 26, 29, 32, 35, 39, 42, 45, 48)
files = []
labels = []
for f in os.listdir(label_dir):
 if f.endswith('.TIF'):
  for row, blur in zip(rows, blurs):
  fname = f.replace('_F1', '_F%d'%blur).replace('_A', '_%s'%row)
  files.append(os.path.join(image_dir, fname))
  labels.append(os.path.join(label_dir, f))
```

```
loader = dc.data.ImageLoader()
dataset = loader.featurize(files, labels)
```

Теперь, когда у нас есть загруженные и обработанные наборы данных, давайте попробуем создать для них несколько моделей глубокого обучения. Разделим этот набор данных на обучающий, проверочный и оценочный наборы, как и раньше:

```
splitter = dc.splits.RandomSplitter()
train_dataset, valid_dataset, test_dataset = splitter.train_valid_test_split(dataset, seed=123)
```

Какую архитектуру мы можем использовать для задачи сегментации изображения? Это уже не вопрос использования простой сверточной архитектуры, поскольку наш вывод сам по себе должен быть изображением (маской сегментации). К счастью для нас, в прошлом была проделана работа над подходящими архитектурами для этой задачи. В архитектуре U-Net используется серия сверток для постепенного *снижения дискретизации* (downsampling), а затем обратного *повышения дискретизации* (upsampling) исходного изображения (рис. 7.14). Эта архитектура эмпирически хорошо справляется с задачей сегментации изображений.

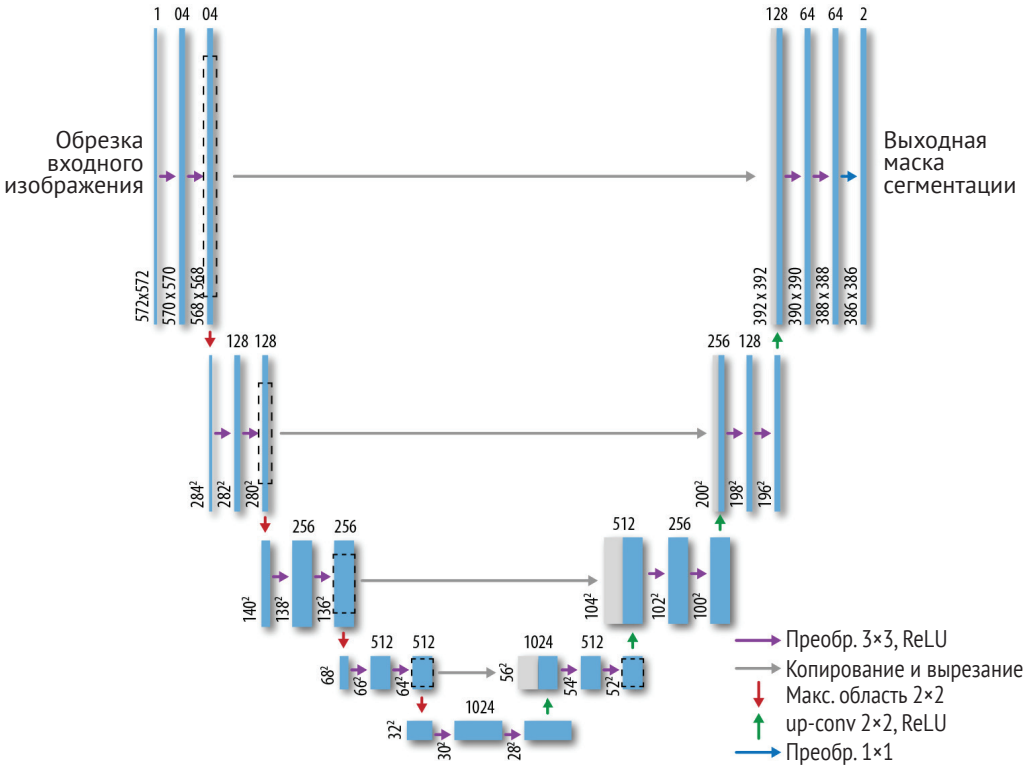


Рис. 7.14 ❖ Представление архитектуры U-Net для сегментации биомедицинских изображений. *Источник*: University of Freiburg (**https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/**)

Давайте теперь внедрим U-Net в DeepChem.

```
learning_rate = dc.models.tensorgraph.optimizers.ExponentialDecay(0.01, 0.9, 250)
model = dc.models.TensorGraph(learning_rate=learning_rate, model_dir='segmentation')
features = layers.Feature(shape=(None, 520, 696, 1)) / 255.0
labels = layers.Label(shape=(None, 520, 696, 1)) / 255.0
# Трехкратное понижение дискретизации.
conv1 = layers.Conv2D(16, kernel_size=5, stride=2, in_layers=features)
conv2 = layers.Conv2D(32, kernel_size=5, stride=2, in_layers=conv1)
conv3 = layers.Conv2D(64, kernel_size=5, stride=2, in_layers=conv2)
# Сделать свертку 1×1.
conv4 = layers.Conv2D(64, kernel_size=1, stride=1, in_layers=conv3)
# Трехкратное повышение дискретизации.
concat1 = layers.Concat(in_layers=[conv3, conv4], axis=3)
deconv1 = layers.Conv2DTranspose(32, kernel_size=5, stride=2, in_layers=concat1)
concat2 = layers.Concat(in_layers=[conv2, deconv1], axis=3)
deconv2 = layers.Conv2DTranspose(16, kernel_size=5, stride=2, in_layers=concat2)
concat3 = layers.Concat(in_layers=[conv1, deconv2], axis=3)
deconv3 = layers.Conv2DTranspose(1, kernel_size=5, stride=2, in_layers=concat3)
# Вычисление окончательного выхода.
concat4 = layers.Concat(in_layers=[features, deconv3], axis=3)
logits = layers.Conv2D(1, kernel_size=5, stride=1, activation_fn=None, in_layers=concat4)
output = layers.Sigmoid(logits)
model.add_output(output)
loss = layers.ReduceSum(layers.SigmoidCrossEntropy(in_layers=(labels, logits)))
model.set_loss(loss)
```

Эта архитектура несколько сложнее, чем для подсчета клеток, но мы используем ту же базовую структуру кода и стек сверточных слоев, как и в предыдущем случае. Как и прежде, давайте протестируем полученную архитектуру с помощью предварительно обученной модели. Инструкции по загрузке предварительно обученной модели доступны в репозитории кода для этой книги. Предварительно скачанные веса можно восстановить, как и раньше:

```
model.restore()
```

Воспользуемся нашей моделью для создания масок сегментации. Вызов `model.predict_on_batch()` позволяет нам спрогнозировать выходную маску для пакета входных данных. Мы можем проверить точность наших прогнозов, сравнивая наши маски с эталонными масками и проверяя долю перекрытия.

```
scores = []
for x, y, w, id in test_dataset.itersamples():
  y_pred = model.predict_on_batch([x]).squeeze()
  scores.append(np.mean((y>0) == (y_pred>0.5)))
print(np.mean(scores))
```

Функция оценки возвращает приблизительно 0,9899, то есть почти 99 % пикселей правильно спрогнозированы! Это хороший результат, но мы должны подчеркнуть, что это была, по сути, детская задачка. Простой алгоритм обработки изображений с порогом яркости, вероятно, сработал бы почти так же хорошо. Но представленные здесь принципы можно перенести на более сложные наборы данных изображений.

Хорошо, теперь, когда мы протестировали предварительно обученную модель, давайте обучим U-Net с нуля в течение 50 эпох и посмотрим, какие результаты мы получим:

```
model.fit(train_dataset, nb_epoch=50, checkpoint_interval=100)
```

Как и раньше, это обучение требует больших вычислительных ресурсов и займет несколько часов на хорошем графическом процессоре. Вряд ли имеет смысл использовать для обучения обычный процессор. Как только эта модель будет обучена, попробуйте запустить ее и посмотреть, что получилось. Удалось ли вам достигнуть точности предварительно обученной модели?

Вычислительные анализы

Подсчет и сегментация клеток являются довольно простыми визуальными задачами, поэтому неудивительно, что модели машинного обучения способны хорошо работать с такими наборами данных. Но ограничивается ли применение глубокого обучения только этими задачами?

К счастью, нет! Машинно обученные модели способны улавливать едва различимые сигналы в наборе данных. Например, одно исследование[1] показало, что глубоко обученные модели способны предсказывать выходные данные флуоресцентных меток из необработанного изображения. Стоит задуматься над тем, как удивителен этот результат! Как мы говорили в разделе о подготовке препаратов, флуоресцентное окрашивание зачастую бывает довольно сложной процедурой. Замечательно, что глубокое обучение помогает избавиться от сложных подготовительных работ.

Бесспорно, это захватывающий результат, но стоит отметить, что это лишь первый шаг. Для широкого применения подобных методов необходимо будет проделать значительную работу.

Заключение

В этой главе мы познакомили вас с основами микроскопии и некоторыми базовыми подходами машинного обучения к системам микроскопии. Предоставили обзорное введение в некоторые фундаментальные вопросы современной микроскопии (особенно в применении к биологическим проблемам), обсудили, где глубокое обучение уже оказало влияние, и предположили области, где оно может оказать еще большее влияние в будущем.

Мы также предоставили подробный обзор некоторых физических и биологических аспектов, связанных с микроскопией. Пытались пояснить вам, почему эта информация очень полезна, даже если вы разработчик автоматизированных конвейеров для обработки микроскопических изображений. Знание физических принципов, таких как дифракционный предел, позволит вам понять, почему используются разные методы микроскопии и насколько глубокое обучение может оказаться критическим для будущего этой отрасли. Знание методов подготовки

[1] *Christensen Eric*. In Silico Labeling: Predicting Fluorescent Labels in Unlabeled Images // **https://github.com/google/in-silico-labeling**.

биологических препаратов поможет вам выявить типы метаданных и аннотаций, которые важно отслеживать при разработке практической системы микроскопии.

Хотя мы очень рады потенциальному применению методов глубокого обучения в микроскопии, для нас важно подчеркнуть, что эти методы имеют ряд ограничений. Например, ряд недавних исследований[1] выявил неустойчивость визуальных сверточных моделей. Простые артефакты могут запутать такие модели и вызвать серьезные проблемы. Например, изображение знака остановки может быть слегка модифицировано, чтобы модель классифицировала его как зеленый сигнал светофора. Подобный сбой модели может стать губительным для беспилотного автомобиля!

Учитывая этот пример, стоит задуматься, каковы потенциальные подводные камни у моделей для глубокой микроскопии? Возможно ли, что модели глубокой микроскопии просто строят правдоподобную комбинацию запомненных предыдущих данных? Даже если это не полное объяснение их результативности, вполне вероятно, что часть силы таких глубоких моделей заключается в *обратном перетекании* (backfilling) сохраненных данных. Это вполне может привести к обнаружению ложных корреляций. При проведении глубокого машинного анализа на основе наборов микроскопических данных будет крайне важно вовремя остановиться и задать вопрос, связаны ли результаты с артефактами модели или с подлинными биологическими явлениями. В следующих главах мы предоставим вам некоторые инструменты для критического исследования моделей, чтобы вы могли достоверно определить, что *на самом деле* выучила ваша модель.

В главе 8 мы углубимся в применение глубокого обучения в медицине и воспользуемся навыками обработки изображений, которые мы рассмотрели в этой главе.

[1] *Rosenfeld Amir, Richard Zemel, and John K. Tsotsos*. The Elephant in the Room. 2018 // **https://arxiv.org/abs/1808.03305**.

Глава 8

Глубокое обучение в медицине

Как мы видели в предыдущей главе, способность извлекать значимую информацию из наборов визуальных данных может успешно применяться в микроскопии. Похожая способность обрабатывать массивы разрозненных данных и делать выводы на их основе востребована и в медицине. Современная медицина нуждается в глубоком анализе обширных результатов медицинских обследований. Инструменты глубокого обучения потенциально могут сделать этот анализ более простым и быстрым (но, возможно, и более трудным для понимания).

Давайте поближе познакомимся с положением дел в медицине. Мы начнем с краткого обзора ранних вычислительных методов и обсудим их ограничения, а затем познакомимся с современным набором методик глубокого обучения в медицинской практике. Мы объясним, как эти новые подходы позволяют нам обойти фундаментальные ограничения старых методов. Завершает главу обсуждение этических соображений насчет применения глубокого обучения в медицине.

Компьютерная диагностика

Разработка компьютерных *диагностических систем* с самого начала была основной целью исследований в области искусственного интеллекта. В самых ранних версиях[1] диагностических систем применялись базы знаний, созданные вручную. В этих системах опытным врачам было предложено записать правила причинно-следственной связи (рис. 8.1).

ЕСЛИ
1) Окраска микроорганизма грамположительная, И (01)
2) Морфологический тип организма – COCCUS, И (02)
3) Подтверждено цепное размножение организмов (03)

ЗНАЧИТ
С предположительной достоверностью (0,7) этот организм является стрептококком. (h1)

Рис. 8.1 ❖ Mycin была ранней экспертной системой, используемой для диагностики бактериальных инфекций. Здесь показан пример правил Mycin для вывода умозаключений. *Источник*: University of Surrey (**http://www.computing.surrey.ac.uk/ai/PROFILE/mycin.html#Certainity%20Factors**)

[1] **https://en.wikipedia.org/wiki/Dendral** или **https://en.wikipedia.org/wiki/Mycin**.

По сути, это были попытки выразить неопределенность через определенные факторы.

Затем правила объединялись с помощью *логического движка* (logical engine). Был разработан ряд эффективных методов вывода умозаключений, которые могли объединять большие базы данных правил. Такие системы традиционно называли экспертными системами.

Что случилось с экспертными системами?

Хотя экспертные системы достигли заметных успехов, их создание потребовало значительных усилий. Правила должны быть тщательно сформулированы экспертами и реализованы квалифицированными инженерами по знаниям. Хотя некоторые экспертные системы достигли поразительных результатов в ограниченных областях, в целом они были слишком ненадежными, чтобы заслуживать обширного применения. Тем не менее экспертные системы оказали сильное влияние на компьютерные науки, и множество современных технологий, включая SQL, XML, байесовские сети и многое другое, черпают вдохновение из технологий экспертных систем.

Если вы разработчик компьютерных технологий, на минутку остановитесь и задумайтесь. Хотя экспертные системы были когда-то ослепительно модной технологией, сегодня они существуют скорее как историческое воспоминание. Весьма вероятно, что большинство современных популярных технологий однажды окажется на свалке истории. Это характерная особенность, а не проблема информатики. На этом поле очень быстро приходят и уходят разные игроки, поэтому мы можем не сомневаться, что новые технологии подарят нам возможности, которые не по силам сегодняшним инструментам. Но в то же время, как и в случае экспертных систем, несомненно, что алгоритмические основы современных технологий будут жить в инструментах завтрашнего дня.

В целом экспертные системы для медицины начинали удачно. Некоторые из них были широко развернуты и приняты на международном уровне[1]. Тем не менее эти системы не смогли добиться необходимого взаимодействия с обычными врачами и медсестрами. Во-первых, такие системы были очень привередливы и сложны в использовании. Во-вторых, они требовали, чтобы их пользователи передавали информацию о пациенте в сложном, строго структурированном формате. Учитывая, что в то время компьютеры едва проникали в рядовые клиники, требования к специализированной подготовке врачей и медсестер оказались слишком завышенными.

Вероятностные диагнозы с байесовскими сетями

Другая серьезная проблема с экспертными системами заключалась в том, что они могли давать только детерминированные прогнозы. Эти детерминированные прогнозы не оставляли много места для неопределенности. Что, если врач осматривал сложного пациента, где диагноз был отнюдь не однозначен? Какое-то время казалось, что если бы экспертные системы удалось модифицировать для учета неопределенностей, этого было бы достаточно для полного успеха экспертных систем.

---

[1] *Asabere Nana Yaw*. mMes: A Mobile Medical Expert System for Health Institutions in Ghana // International Journal of Science and Technology. 2012. № 6 (June). URL: **https://pdfs.semanticscholar.org/ed35/ec162c5916f317162e11e390440bdb1b55b2.pdf**.

Эти представления привели к появлению множества работ по *байесовским сетям* в клинической диагностике. Простой пример байесовской сети показан на рис. 8.2. (Один из авторов этой книги, будучи старшекурсником, провел год в работе над такой системой.) Тем не менее байесовские диагностические системы страдали от прежних ограничений экспертных систем. По-прежнему необходимо было получать от экспертов структурированные знания, и вдобавок разработчики байесовских клинических сетей столкнулись с проблемой получения от врачей осмысленных оценок вероятностей. Эти потребности значительно увеличили накладные расходы на внедрение технологии.

Кроме того, обучение байесовской сети не отличается простотой. Для разных типов байесовских сетей часто приходится использовать разные алгоритмы. Сравните это с алгоритмами глубокого обучения, где методы градиентного спуска работают практически во всех сетях, которые вы можете представить. Простота обучения – вот условие широкого распространения технологии.

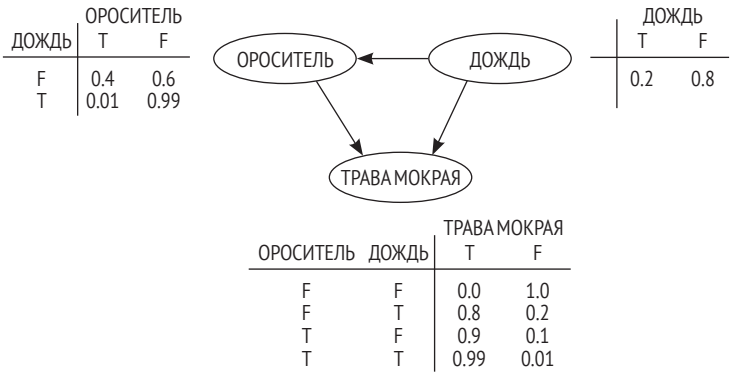


Рис. 8.2 ❖ Простой пример байесовской сети для определения, является ли трава влажной в данном месте.
Источник: **https://commons.wikimedia.org/wiki/File:SimpleBayesNet.svg**

Простота использования способствует распространению

Экспертные системы и байесовские сети не смогли завоевать широкое распространение. По крайней мере, одна из причин этого провала заключалась в том, что обе эти системы были чрезвычайно неудобны для разработчиков. С точки зрения разработчика, при проектировании как байесовской сети, так и экспертной системы необходимо постоянно держать экспертов в курсе разработки. Кроме того, эффективность системы принципиально зависит от способности команды разработчиков извлечь ценные выводы из знаний, предоставленных врачами-экспертами.

С глубокими сетями работать намного проще. Для заданного типа данных (изображения, молекулы, текст и т. д.) и заданной учебной задачи под рукой имеется набор стандартных метрик. Чтобы построить функциональную систему, разработчик должен лишь следовать лучшим статистическим практикам, как показано в этой или многих других книгах. При таком подходе значительно снижается зависимость от экспертных знаний. Этот выигрыш в простоте, без сомнения, стал одной из причин, по которой глубокие сети получили гораздо более широкое распространение.

Данные электронных медицинских карт

Традиционно больницы вели бумажные карты для своих пациентов. В бумажных формах регистрировались результаты анализов, назначенные лекарства, процедуры, что позволяло врачам отслеживать состояние здоровья пациента, быстро взглянув на записи. К сожалению, у бумажных медицинских карт также имелась масса недостатков. Передача записей между больницами требовала значительных усилий. Нелегко было индексировать или искать данные о состоянии здоровья на бумажном носителе.

По этой причине в последние несколько десятилетий в ряде стран произошел почти полный переход от бумажных документов к *электронным медицинским картам* (ЭМК). В США принятие Закона о доступном медицинском обслуживании значительно ускорило внедрение системы ЭМК, и большинство крупных поставщиков медицинских услуг в США в настоящее время хранит свои записи пациентов в стандартной электронной форме.

Разработчики систем машинного обучения буквально набросились на массивы больших данных медицинских карт. На этих данных можно обучить модели, способные предсказывать диагнозы пациентов или потенциальные риски. Во многих отношениях эти основанные на ЭМК модели являются интеллектуальными преемниками экспертных систем и байесовских сетей, о которых мы только что говорили. Как и в случае с более ранними системами, модели ЭМК в основном предназначены для диагностики. Однако более ранние системы стремились помочь врачу поставить диагноз в реальном времени, а новые системы в основном работают «задним числом», на основе накопленных данных.

Несмотря на некоторые заметные успехи отдельных проектов, изучение данных ЭМК остается сложной практической задачей. Из-за проблем с конфиденциальностью не существует большого количества доступных открытых наборов данных. В результате до сих пор только небольшая группа избранных исследователей была допущена к проектированию систем машинного обучения на ЭМК. Кроме того, данные ЭМК имеют тенденцию быть очень грязными. Поскольку врачи и медсестры вводят информацию вручную, большинство наборов данных страдает от пропусков полей и всевозможных неформальных соглашений. Создание надежных моделей, которые работают с отсутствующими данными, оказалось сложной задачей.

Коды ICD-10

Стандарт ICD-10 (International Classification of Diseases, в России МКБ-10 – международная классификация болезней) представляет собой набор «кодов» заболеваний и симптомов у пациентов. Этот стандарт в последние годы получил широкое применение, поскольку он позволяет страховщикам и правительственным учреждениям согласовывать стандартные методы и стоимость лечения.

Коды ICD-10 «квантуют» (делают дискретным) многомерное непрерывное пространство человеческих заболеваний. Стандартизируя, они позволяют врачам сравнивать и группировать пациентов. Стоит отметить, что по этой причине такие коды, вероятно, окажутся актуальными для разработчиков систем и моделей ЭМК. Если вы проектируете хранилище данных для новой системы ЭМК, обязательно подумайте, в каком месте там будут храниться коды!

Форматы FHIR

Ресурсы оперативного взаимодействия в здравоохранении (Fast Healthcare Interoperability Resources, FHIR) – это структура данных, которая была разработана для представления клинических данных в стандартном и гибком формате[1]. Недавняя работа Google[2] показала, как необработанные данные ЭМК могут автоматически преобразовываться в формат FHIR. Использование FHIR в качестве общего формата означает, что можно разработать стандартные глубокие архитектуры, которые будут применимы к произвольным данным ЭМК. Следовательно, для таких форматов можно будет создать инструменты с открытым исходным кодом, действующие по принципу «подключи и работай». Эта работа все еще находится в начале пути, но обещает захватывающий прогресс в научной медицине. Хотя стандартизация на первый взгляд кажется скучным занятием, она закладывает основу для будущих достижений, потому что любой желающий сможет продуктивно работать с большими данными.

Однако ситуация с хаосом данных ЭМК постепенно начинает исправляться. Появились улучшенные инструменты, как для предварительной обработки данных, так и для обучения моделей на данных ЭМК. Например, система DeepPatient обучает *шумоподавляющий автокодер* на медицинских картах пациентов, чтобы создать представление пациента, которое затем используется для прогнозирования диагноза и перспектив лечения[3]. В системе запись пациента преобразуется из набора неупорядоченной текстовой информации в вектор. Эта стратегия преобразования разнородных типов данных в векторы хорошо зарекомендовала себя в глубоком обучении, и, похоже, она обещает существенные улучшения в системах ЭМК. В литературе появились описания различных моделей, основанных на данных ЭМК; многие из этих моделей используют новейшие инструменты глубокого обучения, такие как рекуррентные сети или обучение с подкреплением. Глядя на эти модели, можно представить направление развития отрасли на ближайшие несколько лет.

Возможно ли неконтролируемое обучение?

На протяжении большей части этой книги мы в первую очередь демонстрировали методы *контролируемого обучения* (supervised learning). Существует целый класс методов *неконтролируемого обучения* (unsupervised learning), которые не имеют такой же зависимости от предварительно размеченных обучающих данных. Давайте на минуту остановимся, поскольку мы до сих пор не ввели понятие неконтролируемое обучение. Основная идея заключается в том, что у нас больше нет меток, связанных с точками данных. Например, представьте, что у нас есть набор записей ЭМК (то есть входной вектор), но нет данных о результатах лечения пациентов. Нам не с чем сравнивать вывод модели. Что мы можем сделать?

Самый простой ответ – мы можем *кластеризовать* записи. Для примера, представьте, что у нас есть «близнецы», у которых записи ЭМК идентичны. Наверное,

---

[1] *Mandel JC, et al*. SMART on FHIR: A Standards-Based, Interoperable Apps Platform for Electronic Health Records. 2016 // **https://doi.org/10.1093/jamia/ocv189**.

[2] *Rajkomar Alvin, et al*. Scalable and Accurate Deep Learning with Electronic Health Records // NPJ Digital Medicine. 2018. URL: **https://arxiv.org/pdf/1801.07860.pdf**.

[3] *Miotto Riccardo, Li Li, Brian A. Kidd, and Joel T. Dudley*. Deep Patient: An Unsupervised Representation to Predict the Future of Patients from the Electronic Health Records. 2016 // **https://doi.org/10.1038/srep26094**.

будет разумно предположить, что результаты этих двух пациентов похожи. Неконтролируемые методы обучения, такие как k-среднее или автоэнкодеры, реализуют более сложные формы механизма предположений. Вы увидите сложный пример неконтролируемого алгоритма позже в главе 9.

Иногда неконтролируемые методы убедительно выявляют скрытые закономерности, а иногда кажется, что они действуют наобум. Хотя у нас есть примеры успешных систем, таких как DeepPatient, в целом неконтролируемые методы обучения все еще слишком капризны и нестабильны, чтобы получить широкое распространение. Для исследователей получение стабильных результатов в системах с неконтролируемым обучением пока остается сложной и неизбежной проблемой.

В чем опасность больших баз данных ЭМК пациентов?

Ряд крупных учреждений стремится к включению всех своих пациентов в системы ЭМК. Что произойдет, когда эти огромные наборы данных стандартизируются (возможно, в таком формате, как FHIR) и станут взаимно совместимыми? С одной стороны, появится возможность создавать полезные приложения, например поиск пациентов с фенотипом определенного заболевания. Такой целенаправленный поиск может помочь врачам более эффективно подбирать лечение, особенно для пациентов с редкими заболеваниями.

Однако не нужно много воображения, чтобы представить злонамеренное использование больших баз данных пациентов. Например, страховщики могут использовать системы предсказания заболеваний, чтобы заранее отказать в страховании пациентам с более высоким риском. Или ведущие хирурги, стремящиеся поддерживать высокие показатели выживаемости пациентов, могут избегать пациентов, которым система присвоила высокий уровень риска. Как нам защититься от таких опасностей?

Многие из вопросов, которые поднимают системы машинного обучения, не могут быть решены с помощью инструментов машинного обучения. Скорее всего, ответы на эти вопросы будут зависеть от законодательства, которое должно запретить хищническое поведение врачей и страховщиков или других лиц.

Действительно ли ЭМК помогают врачам?

Хотя ЭМК, несомненно, помогают в разработке алгоритмов обучения, вовсе не столь очевидно, что ЭМК действительно облегчают жизнь врачей. Отчасти проблема заключается в том, что современные ЭМК нуждаются в ручном вводе данных со стороны врачей. Для пациентов это новая история на старый лад: раньше врач тратил большую часть консультации на ручные записи в истории болезни, а теперь он тратит то же самое время, глядя на дисплей, а не на реального пациента.

Такое положение дел оставляет несчастными как пациентов, так и врачей[1]. Врачи чувствуют усталость, так как большую часть своего времени они расходуют на ввод канцелярских данных, а не на лечение пациентов. Пациенты, в свою очередь, чувствуют, что врачи игнорируют их. Остается лишь надеяться, что будущие приложения, основанные на глубоком обучении, улучшат положение дел.

Но обратите внимание, что и следующее поколение инструментов глубокого обучения может оказаться столь же недружественным и бесполезным для врачей. Ведь разработчики ЭМК тоже не планировали создавать недружественные системы.

[1] *Gawande Atul*. Why Doctors Hate Their Computers // The New Yorker. 2018. URL: **https://www.newyorker.com/magazine/2018/11/12/why-doctors-hate-their-computers**.

Глубокая радиология

Радиология – это наука о медицинском сканировании организма для диагностики заболеваний. Врачи используют различные технологии, такие как *МРТ* (магниторезонансная томография), *УЗИ* (ультразвуковое исследование), *рентгеновские снимки* и *КТ* (компьютерная томография). Общее назначение этих технологий состоит в том, чтобы диагностировать состояние пациента, исходя из полученного изображения. Пожалуй, эта задача идеально подходит для методов сверточного обучения. Как мы видели в предыдущих главах, методы глубокого обучения способны вырабатывать очень сложные функции на данных из изображений. Большая часть современной радиологии (по крайней мере, техническая часть) заключается в классификации и обработке сложных данных медицинских изображений.

В этом разделе мы обзорно представим несколько различных типов аппаратных исследований и кратко рассмотрим некоторые приложения для глубокого обучения. Многие из этих приложений качественно похожи. Они начинают с получения достаточно большого набора изображений (сканов) из медицинского учреждения. Далее эти сканы применяются для обучения сверточной архитектуры (рис. 8.3). Как правило, архитектура представляет собой стандартную архитектуру VGG или ResNet, но иногда с некоторыми изменениями в структуре ядра. Обученная модель часто (по крайней мере, если верить простой статистике) демонстрирует хорошие результаты в соответствующей медицинской задаче.

Достижения глубокого обучения привели к далеко идущим, возможно, завышенным ожиданиям. Некоторые видные ученые в области искусственного интеллекта, в частности Джефф Хинтон (Geoff Hinton), отметили[1], что глубокое обучение в радиологии продвинется настолько далеко, что больше не имеет смысла готовить новых радиологов в ближайшем будущем. Это действительно так? В последнее время появилось множество применений, в которых системы глубокого обучения достигли того, что похоже на производительность, близкую к человеческой. Тем не менее эти достижения сопровождаются множеством оговорок, а системы часто оказываются неустойчивыми в незнакомых ситуациях.

Наше мнение таково, что вероятность прямой замены врачей «один к одному» остается низкой, но существует реальный риск систематического смещения. Что это значит? Новые стартапы работают над созданием бизнес-моделей, в которых системы глубокого обучения выполняют большую часть анализа изображений, и остается лишь несколько докторов, принимающих окончательное решение.

Обучается ли глубокое обучение медицине?

Настало время задать очень серьезный вопрос: *чему на самом деле* глубокие модели учатся на медицинских изображениях? К сожалению, похоже, что во многих случаях глубокие модели преуспевают в выявлении *немедицинских факторов* в изображениях. Например, модель может неявно научиться определять медицинский центр, в котором был сделан снимок. Поскольку одни центры чаще используются для лечения более серьезных заболеваний, а другие занимаются менее сложными пациентами, может получиться так, что модель будет ставить диагноз или давать прогноз, исходя из технических особенностей снимков конкретного медицинского центра, а не фактического состояния пациента, хотя внешне это будет выглядеть как вполне достоверный медицинский анализ.

[1] AI, Radiology and the Future of Work // The Economist. 2018. URL: **https://econ.st/2HrRDuz**.

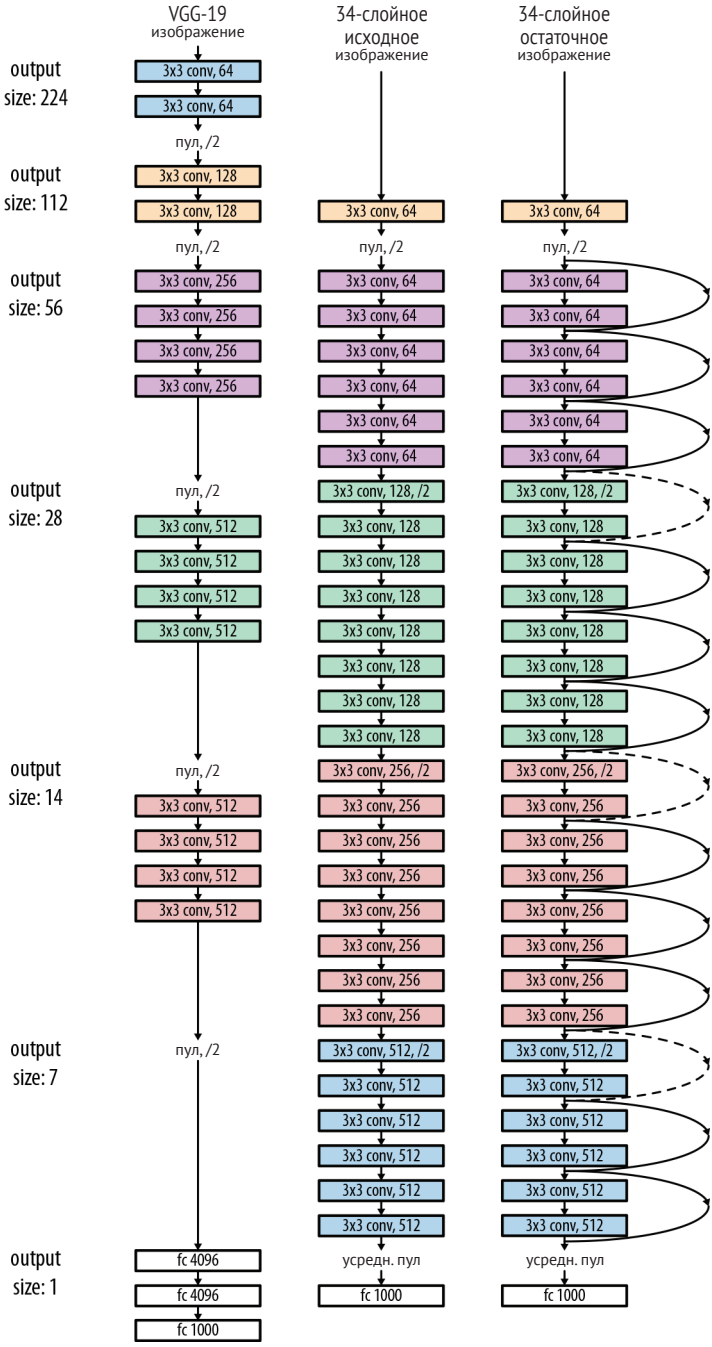


Рис. 8.3 ❖ На этой диаграмме показаны некоторые стандартные сверточные архитектуры (VGG-19, Resnet-34, число указывает количество примененных сверток). Эти архитектуры являются стандартными для работы с изображениями и обычно используются для медицинских приложений

Что можно сделать в таких случаях? В литературе все еще нет ответа на этот вопрос, но есть пара перспективных идей. Во-первых, предложено использовать развивающуюся методику интерпретации моделей, чтобы тщательно изучить то, чему учится модель. В главе 10 мы рассмотрим ряд методов интерпретации моделей.

Другой подход заключается в проведении *проспективных испытаний* с использованием модели в клинике. Проспективные испытания остаются золотым стандартом при тестировании медицинских вмешательств, и, вероятно, они останутся таковыми и для методов глубокого обучения.

Рентгенография и компьютерная томография

Рентгенография (*радиография*, если говорить точно) означает использование рентгеновских лучей, чтобы рассмотреть какую-то внутреннюю структуру тела (рис. 8.4). Компьютерная томография представляет собой вариант рентгенографии, при котором источник рентгеновского излучения и детекторы вращаются вокруг исследуемого объекта, позволяя получать трехмерные изображения.

Рис. 8.4 ❖ Первый медицинский рентгеновский снимок, сделанный Вильгельмом Рентгеном – изображение руки его жены Анны Берты Людвиг. Рентгенография прошла долгий путь со времени этого снимка, и есть шанс, что глубокое обучение станет следующим шагом!

Существует распространенное заблуждение, что рентгеновские снимки способны отображать только «твердые» объекты, такие как кости. Это не так. Компьютерная томография обычно используется для визуализации мягких тканей тела, таких как мозг (рис. 8.5). Фактически вариант рентгеновского сканирования (рентгеновское излучение с обратным рассеянием) часто используется в аэро-

портах для досмотра путешественников на контрольно-пропускных пунктах. Для получения *маммограммы* (снимка тканей молочной железы) также используют низкоэнергетическое рентгеновское излучение.

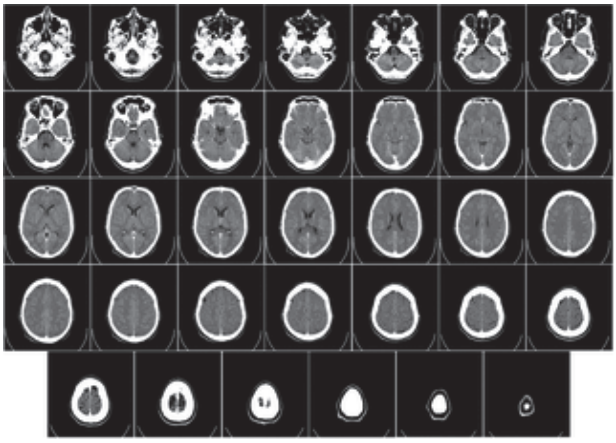


Рис. 8.5 ❖ КТ-сканирование человеческого мозга снизу вверх. Обратите внимание на способность компьютерной томографии предоставлять трехмерную информацию. *Источник*: **https://commons.wikimedia.org/wiki/File:Computed_tomography_of_human_brain_-_large.png**

Стоит отметить, что рентгеновское излучение может провоцировать рак, поэтому общая цель рентгенографии состоит в том, чтобы свести к минимуму облучение пациентов. Эта проблема актуальна для компьютерных томографов, где пациент подвергается длительному воздействию излучения при сборе необходимого объема данных. Для уменьшения количества сканирующих проходов применяются различные алгоритмы обработки сигналов. Недавние научные работы продемонстрировали впечатляющие результаты по использованию глубокого обучения для восстановления изображения, так что есть шанс сделать облучение еще меньше.

Тем не менее чаще всего глубокое обучение в медицине используется для классификации изображений. Например, сверточные сети были использованы для классификации стадий болезни Альцгеймера по компьютерным томограммам головного мозга[1]. В других работах была продемонстрирована возможность диагностировать пневмонию по рентгеновским снимкам грудной клетки почти с врачебной точностью[2]. Глубокое обучение также продемонстрировало высокую точность классификации в маммографии[3].

---

[1] *Gao Xiaohong W., Rui Hui, and Zengmin Tian*. Classification of CT Brain Images Based on Deep Learning Networks. 2017 // **https://doi.org/10.1016/j.cmpb.2016.10.007**.

[2] *Pranav Rajpurkar et al*. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. 2017 // **https://arxiv.org/pdf/1711.05225.pdf**.

[3] *Ribli Dezso et al*. Detecting and Classifying Lesions in Mammograms with Deep Learning. 2018 // **https://doi.org/10.1038/s41598-018-22437-z**.

Достичь человеческого уровня точности очень сложно!

Когда в статье говорится, что система достигает почти человеческой точности, стоит остановиться и обдумать, что это значит. Обычно авторы статьи выбирают какую-то метрику (скажем, ROC-AUC). Затем группа независимых врачей работает, чтобы аннотировать (разметить) выбранный набор тестов для исследования. Точность модели в этом тестовом наборе сравнивается со «средним» врачом (часто средним или медианным из оценок врача). Это довольно сложный процесс, и существует несколько причин, по которым это оценочное сравнение может пойти не так. Во-первых, может сыграть свою роль выбор метрики. Слишком часто случается так, что изменение метрики приводит к различиям в оценке. Хороший анализ будет учитывать несколько различных метрик, чтобы убедиться, что общая оценка устойчива к различиям в метрике.

Второй момент, на который стоит обратить внимание, заключается в том, что существуют значительные различия между самими врачами. Стоит проверить правильность выбора вашего «среднего» врача. Самым надежным показателем качества здесь может быть вопрос о том, способен ли ваш алгоритм победить «лучшего» доктора в группе.

Третья тонкость заключается в том, что крайне сложно убедиться, что набор тестов не «загрязнен» (см. предыдущее предупреждение об артефактах обучения в глубоких сетях). Могут возникнуть и более скрытые формы загрязнения, при которых сканы одного и того же пациента могут случайно оказаться как в тренировочном, так и в тестовом наборе. Если ваша модель имеет очень высокую точность, стоит провести двойную и тройную проверку на наличие таких утечек. Все мы когда-то допускали подобные ошибки при подготовке данных.

И наконец, сама по себе «точность человеческого уровня» мало что значит. Как мы уже отмечали ранее, некоторые экспертные системы и байесовские сети достигли точности человеческого уровня при выполнении ограниченных задач, но не смогли получить широкое распространение в медицине. Причина этого заключается в том, что врачи решают целый *комплекс задач*, которые тесно связаны между собой. Определенный врач может проиграть глубокой модели при анализе изображения, но предложить гораздо лучшую диагностику на основе другой информации. Стоит помнить, что задачи диагностики часто носят синтетический характер и не всегда раскрывают все возможности врача. Для более точной оценки эффективности новых методов необходимы проспективные клинические испытания с использованием систем глубокого обучения в режиме реального времени.

Гистология

Гистология – это исследование тканей организма, часто с помощью микроскопа. Мы не будем много говорить о гистологии, поскольку проблемы, с которыми сталкиваются разработчики систем глубокой гистологии, являются подмножеством проблем, с которыми сталкивается глубокая микроскопия. Мы говорили об этом в главе 7, а здесь просто отметим, что модели глубокого обучения добились высоких результатов в гистологических исследованиях.

Магниторезонансная томография

Магниторезонансная томография (МРТ) является еще одной формой радиологии, широко применяемой в медицине. В отличие от рентгеновского сканирования, МРТ использует для визуализации сильные магнитные поля. Одним из преимуществ МРТ является ограниченное облучение. Тем не менее МРТ-сканирование часто требует, чтобы пациенты долго лежали в шумной и тесной МРТ-аппаратуре, что может быть гораздо более неприятным для пациентов, чем рентгеновский снимок.

Как и компьютерная томография, МРТ способна собирать трехмерные изображения. В этой области тоже ведутся исследования по реконструкции объемных изображений при помощи алгоритмов глубокого обучения. В некоторых ранних исследованиях утверждается, что методы глубокого обучения способны улучшить традиционные методы обработки сигналов при восстановлении изображений МРТ с уменьшенным временем сканирования. Кроме того, в ряде исследований были предприняты попытки использовать глубокие сети для классификации, сегментации и обработки изображений МРТ. Некоторые исследователи добились значительных успехов.

Глубокое обучение в обработке сигналов

Мы упоминали, что глубокие сети применяются для более эффективной реконструкции изображений в КТ и МРТ. Эти приложения отражают общую тенденцию использования глубокого обучения в обработке сигналов. Методы глубокого обучения для сверхразрешающей микроскопии также относятся к этому направлению.

Работы по совершенствованию методов обработки сигналов очень интересны с фундаментальной точки зрения. Ведь обработка сигналов является строго математической и глубоко проработанной областью науки. Тот факт, что глубокое обучение предлагает новые направления, сам по себе является удивительной новостью! Однако стоит отметить, что у традиционных алгоритмов обработки сигналов сейчас очень сильные начальные позиции. Поэтому, в отличие от классификации изображений, в обработке сигналов глубокие методы пока не предлагают прорывных улучшений точности. Тем не менее это область активных исследований. Будет неудивительно, если работы по глубокой обработке сигналов в конечном итоге окажутся даже более востребованы, чем простая обработка изображений, в связи с очень широким спектром применения таких методов.

Стоит отметить, что врачи используют множество других аппаратных методов исследования организма. Учитывая взрыв интереса к приложениям глубокого обучения, основанным на мощных инструментах с открытым исходным кодом, можно поспорить, что для любого типа медицинских исследований найдется одна или две работы, пытающиеся применить глубокое обучение для этой задачи. Например, глубокое обучение было применено к ультразвуку, расшифровке электрокардиограммы (ЭКГ), обнаружению рака кожи и многому другому.

Сверточные сети являются чрезвычайно мощным инструментом, так как большая часть человеческой деятельности вращается вокруг обработки сложной визуальной информации. Кроме того, рост числа фреймворков с открытым исходным кодом означает, что исследователи во всем мире присоединились к гонке по применению методов глубокого обучения к новым типам изображений. Этот тип исследований хорош тем, что в нем можно без особого труда применять наборы стандартных инструментов и обходиться относительно скромными вычислительными ресурсами. Если вы читаете данную книгу, будучи сотрудником медицинской компании, то именно эти свойства глубокого обучения делают его интересным для вас как для практикующего врача.

Модель глубокого обучения в качестве лечебного средства

До сих пор мы говорили о том, что обучаемые модели могут быть эффективными помощниками врачей в процессах диагностики заболеваний и расшифровки анализов. Тем не менее есть убедительные доказательства того, что модели с глу-

боким обучением могут превратиться из помощников врачей в самостоятельное лечебное средство.

Как такое возможно? Одна из величайших возможностей глубокого обучения заключается в том, что впервые стало возможным создание программного обеспечения, работающего непосредственно с данными восприятия. Системы машинного обучения могут потенциально служить «глазами» и «ушами» для пациентов с ограниченными возможностями. Зрительная система способна помочь пациентам с нарушениями зрения более эффективно ориентироваться в мире. Система обработки звука аналогично может помочь пациентам с нарушениями слуха. Эти системы сталкиваются с рядом специфических проблем, потому что они должны эффективно работать в режиме реального времени. Вдобавок все модели, которые мы рассмотрели в этой книге, были пакетными системами для развертывания на относительно мощной локальной машине, а не моделью, подходящей для развертывания на встроенном или мобильном устройстве. Существует целый ряд проблем, связанных с машинным обучением в потребительском оборудовании, которые мы не будем здесь рассматривать, но мы призываем заинтересованных читателей углубиться в предмет.

Интересно отметить, что, оказывается, существует отдельный класс *программно ориентированной терапии*, в которой используются мощные эффекты воздействия современного программного обеспечения на человеческий мозг. Недавние исследования показали, что современные программные приложения, такие как Facebook, Google, WeChat и им подобные, могут вызывать сильную зависимость. Эти приложения используют яркие цвета и дизайн, воздействующие на многие из тех же центров нашего мозга, что и игровые автоматы казино. Все больше исследователей признают, что цифровая зависимость[1] – это реальная проблема, с которой сталкиваются многие пациенты. Это обширная область исследований, выходящая за рамки данной книги, но мы отмечаем, что уже разработаны приложения, использующие направленные психологические эффекты в качестве терапии для пациентов, борющихся с депрессией и другими расстройствами психики.

Диабетическая ретинопатия

До сих пор в этой главе мы обсуждали теоретические соображения относительно применения глубокого обучения в медицине. А сейчас мы закатаем рукава и возьмемся за практику. В частности, мы создадим модель, помогающую диагностировать прогрессирующую диабетическую ретинопатию.

Диабетическая ретинопатия (diabetic retinopathy) – это заболевание, при котором здоровье глаз нарушается из-за диабета. Это основная причина слепоты, особенно в развивающихся странах. *Глазное дно* (fundus) – это внутренняя область глаза, противоположная хрусталику. Общепринятая диагностика диабетической ретинопатии заключается в том, что врачи просматривают изображение глазного дна пациента и делают выводы о наличии и степени развития ретинопатии. Технология съемки глазного дна пациента уже хорошо отработана (рис. 8.6).

[1] **https://en.wikipedia.org/wiki/Digital_addict**.

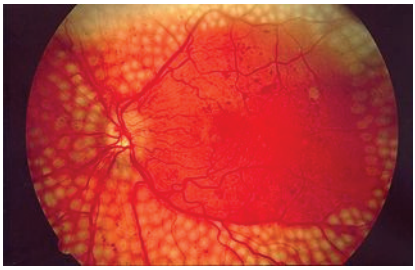


Рис. 8.6 ❖ Изображение глазного дна пациента, который подвергся лечению рассеянным лазерным излучением. *Источник*: **https://commons.wikimedia.org/wiki/File:Fundus_photo_showing_scatter_laser_surgery_for_diabetic_retinopathy_EDA09.JPG**

Задача глубокого обучения применительно к диабетической ретинопатии заключается в разработке алгоритма, который мог бы классифицировать степень заболевания пациента по изображению его глазного дна. В настоящее время для таких прогнозов требуются квалифицированные врачи или техники, но у нас есть все основания надеяться, что система машинного обучения сможет точно предсказывать прогресс заболевания по изображениям глазного дна пациента. Пациенты могли бы использовать дешевый метод оценки риска, прежде чем обратиться за консультацией к более дорогому врачу-специалисту для постановки окончательного диагноза.

Кроме того, в отличие от данных ЭМК, изображения глазного дна почти не содержат конфиденциальной информации о пациентах, что облегчает формирование больших обучающих наборов изображений глазного дна. По этим причинам был проведен ряд исследований в области машинного обучения и наборов данных диабетической ретинопатии. В частности, Kaggle спонсировал конкурс[1], нацеленный на создание хороших моделей диабетической ретинопатии, и подготовил набор изображений глазного дна высокого разрешения. В оставшейся части этого раздела вы узнаете, как использовать DeepChem для построения классификатора диабетической ретинопатии на наборе данных Kaggle DR.

Получение набора данных о диабетической ретинопатии Kaggle

Условия конкурса Kaggle запрещают нам размещать данные непосредственно на серверах DeepChem. По этой причине вам нужно будет загрузить данные собственноручно с сайта Kaggle. Вам придется зарегистрировать учетную запись в Kaggle и загрузить набор данных через их API. Полный набор данных довольно большой (80 ГБ), поэтому вы можете загрузить подмножество данных, если ваше интернет-соединение ограничивает трафик.

Обратитесь к справочному материалу, сопровождающему эту книгу, для получения инструкций о загрузке набора данных. Функции загрузки изображения требуют, чтобы обучающие данные были размещены в определенной структуре каталогов. Подробные сведения об этом формате каталога доступны в репозитории GitHub.

Первым шагом для работы с этими данными является предварительная обработка и загрузка сырых данных. В частности, мы обрезаем каждое изображение,

[1] **https://www.kaggle.com/c/diabetic-retinopathy-detection**.

чтобы сфокусироваться на его центральном квадрате, содержащем сетчатку. Затем мы масштабируем этот центральный квадрат к размеру 512×512.

Работа с изображениями высокой четкости

Многие наборы медицинских и научных данных содержат изображения с очень высоким разрешением. Хотя очень заманчиво обучать модели непосредственно на исходных изображениях, обычно это сложно сделать с вычислительной точки зрения. Во-первых, большинство современных графических процессоров имеет ограниченную память. Это означает, что обучение моделей с очень высоким разрешением может оказаться невозможным на стандартном оборудовании. Кроме того, большинство систем обработки изображений (на данный момент) ожидает, что их входные изображения будут иметь фиксированный размер. Это означает, что изображения с высоким разрешением с разных камер должны быть обрезаны и масштабированы, чтобы соответствовать стандартному размеру.

К счастью, оказывается, что обрезка и изменение размера изображений обычно не наносят серьезного ущерба качеству систем машинного обучения. Также обычно практикуется множественное *дополнение данных* (data augmentation), при котором из каждого исходного изображения автоматически генерируется несколько *измененных дополняющих изображений* (perturbed images). В нашем примере мы выполняем несколько стандартных дополнений данных. Мы рекомендуем вам изучить код, выполняющий дополнения, поскольку он может оказаться полезным инструментом для ваших собственных проектов.

Основные данные хранятся в наборе каталогов на диске. Для загрузки изображений с диска мы используем класс DeepChem `ImageLoader`. Если вам интересно, можете подробно изучить этот код загрузки и предварительной обработки, но мы завернули его в удобную вспомогательную функцию. Продолжая традицию загрузчиков MoleculeNet, эта функция также выполняет случайное разделение данных на обучающий, проверочный и оценочный наборы:

```
train, valid, test = load_images_DR(split='random', seed=123)
```

Теперь, когда у нас есть обучающие данные, давайте построим сверточную архитектуру для обучения на этом наборе данных. Архитектура для этой задачи довольно стандартна и напоминает другие архитектуры, которые вы уже видели в этой книге, поэтому мы не будем ее здесь повторять. Вот вызов обертки объекта для базовой сверточной сети:

```
# Определяем и строим модель
model = DRModel(
    n_init_kernel=32,
    batch_size=32,
    learning_rate=1e-5,
    augment=True,
    model_dir='./test_model')
```

Этот пример кода определяет в DeepChem сверточную сеть диабетической ретинопатии. Как мы увидим позже, обучение этой модели потребует некоторых сложных вычислений. По этой причине мы рекомендуем вам загрузить нашу предварительно обученную модель с веб-сайта DeepChem и использовать ее для ознакомительного изучения. Мы уже обучили эту модель на полном наборе данных диабетической ретинопатии Kaggle и для вашего удобства сохранили ее веса. Вы можете использовать следующие команды для загрузки и сохранения модели (обратите внимание, что первую команду следует вводить в одной строке, без пробелов в адресе):

```
wget https://s3-us-west-1.amazonaws.com/deepchem.io/featurized_datasets/DR_model.tar.gz
mv DR_model.tar.gz test_model/
cd test_model
tar -zxvf DR_model.tar.gz
cd ..
```

Затем вы можете восстановить веса обученной модели следующим образом:

```
model.build()
model.restore(checkpoint="./test_model/model-84384")
```

Мы восстанавливаем конкретную «контрольную точку» предварительно обученной модели. Более подробная информация о процессе восстановления контрольной точки и полных сценариях восстановления приведена в репозитории кода этой книги на GitHub. Располагая предварительно обученной моделью, давайте вычислим для нее некоторую базовую статистику:

```
metrics = [
    dc.metrics.Metric(DRAccuracy, mode='classification'),
    dc.metrics.Metric(QuadWeightedKappa, mode='classification')
]
```

Существует ряд показателей, которые полезны для оценки моделей диабетической ретинопатии. Здесь `DRAccuracy` – это просто точность модели (процент правильных меток), а `QuadWeightedKappa` – это так называемая *каппа Коэна* (Cohen's Kappa), статистический показатель, который используется для измерения взаимного согласия между двумя классификаторами (или экспертами). Это полезно, так как проблема диабетической ретинопатии относится к задачам *многоклассовой классификации* (multiclass classification).

Давайте оценим нашу предварительно обученную модель на тестовом наборе с нашими метриками:

```
>> model.evaluate(test, metrics)

computed_metrics: [0.9339595787076572]
computed_metrics: [0.8494075470551462]
```

Базовая модель имеет точность 93,4 % на оценочном наборе. Неплохо! (Важно отметить, что это не то же самое, что оценочный набор Kaggle. Мы ведь просто разделили для нашего эксперимента обучающий набор Kaggle на три части – обучающую, проверочную и оценочную. Вы можете попробовать отправить свою обученную модель в Kaggle, чтобы они оценили ее на своем оценочном наборе.) А что, если вы заинтересованы в обучении полной модели с нуля? Обучение на хорошем графическом процессоре займет один-два дня, но в целом это делается достаточно просто:

```
for i in range(10):
    model.fit(train, nb_epoch=10)
    model.evaluate(train, metrics)
    model.evaluate(valid, metrics)
    model.evaluate(valid, cm)
    model.evaluate(test, metrics)
    model.evaluate(test, cm)
```

Мы обучаем модель в течение 100 эпох, периодически останавливаясь, чтобы распечатать результаты обучения. Если вы собираетесь выполнить эту работу, убедитесь, что ваша машина не выключится или не перейдет в режим сна на полпути. Нет ничего более раздражающего, чем потерять сделанное из-за появления экранной заставки!

Перспективы глубокого обучения в медицине

Во многих отношениях применение машинного обучения в медицине может оказать большее влияние на нашу жизнь, чем применение в других областях, о которых мы говорили раньше. Какие-то другие применения глубоких моделей, возможно, изменили или изменят то, что вы делаете на работе, но системы здравоохранения с машинным обучением скоро изменят ваш личный опыт в сфере здравоохранения наряду с опытом миллионов и миллиардов других людей. По этой причине стоит остановиться и обдумать некоторые этические последствия.

Этические соображения

В обозримом будущем обучающие данные для медицинских систем, вероятно, будут предвзятыми. Скорее всего, обучающие данные будут получены из медицинских систем развитых стран. В результате построенные модели могут оказаться значительно менее точными в тех частях мира, где в настоящее время отсутствуют развитые медицинские системы.

Кроме того, сбор данных о пациентах сам по себе чреват потенциальными этическими проблемами. Рассмотрим случай с Генриеттой Лакс (Henrietta Lacks). Врач г-жи Лакс получил образец ее опухоли без ее согласия. Клеточная линия, культивируемая из опухоли г-жи Лакс, знаменитая в научном мире «HeLa», стала стандартным биологическим инструментом и использовалась в тысячах научных работ. Ни один цент из доходов от этих исследований никогда не доходил до семьи мисс Лакс. Генриетта Лакс была афроамериканской пациенткой в Балтиморе 1950-х годов, которую лечил белый врач, не посчитавший нужным сообщить семье о взятых им пробах. Семья г-жи Лакс не подозревала о клеточной линии «HeLa» до 1970-х годов, когда с ними связались медицинские исследователи, стремящиеся получить дополнительные образцы.

Как эта ситуация может повториться в эпоху глубокого обучения? Медицинские карты пациента могут быть использованы для машинного обучения без согласия пациента или его семьи. Или, что более реалистично, пациент либо его семья будут вынуждены подписать согласие на использование данных у постели больного в надежде на излечение в последнюю минуту.

Эти сценарии вызывают тревогу. Никто из нас не хочет, чтобы права наших любимых членов семьи были нарушены государственной медициной или коммерческими стартапами. Как мы можем предотвратить этические нарушения? Если вы участвуете в сборе данных, сделайте паузу и задайте вопрос, откуда поступают данные. Все ли соответствующие законы соблюдаются надлежащим образом? Если вы работаете ученым или разработчиком в компании или исследовательском институте, значит, у вас есть ценные навыки, в которых заинтересована ваша организация. Если вы остановитесь и начнете задавать вопросы, то и ваши коллеги могут поступить так же. И если ваша организация отказывается вас слу-

шать, благодаря вашим навыкам вы можете найти работу в другой организации, которая придерживается более высоких этических стандартов.

Потеря работы

Большинство областей, рассматриваемых в других главах этой книги, – относительно нишевые научные дисциплины. Таким образом, даже при больших успехах машинного обучения этим областям не грозит безработица. Скорее, наоборот, следует ожидать роста занятости, поскольку нишевые предметы внезапно станут доступны для более широкого круга разработчиков и ученых.

Совершенно иная ситуация в медицине и здравоохранении. Здравоохранение, с миллионами врачей, медсестер, техников и прочего персонала, является одной из крупнейших отраслей в мире. Что произойдет, когда значительная часть этой рабочей силы столкнется с конкуренцией в виде инструментов глубокого обучения?

В основе большей части современной медицины лежит человеческая деятельность. Наличие надежного врача первой помощи, которому можно доверять и который заботится об интересах больного, имеет огромное значение для пациента. Вполне возможно, что автоматизация рутинной работы врачей могла бы улучшить качество медицинского обслуживания.

Реформа здравоохранения США в 2010 году ускорила повсеместное внедрение электронных медицинских карт в американскую медицину. Многие врачи утверждают, что система медицинских карт глубоко недружелюбна и требует множества ненужных административных действий. Частично это происходит из-за плохо продуманного программного обеспечения, усугубленного обилием норм и правил. Но отчасти это связано и с ограниченными возможностями современных методов обработки данных. Системы глубокого обучения могут снизить нагрузку на врачей, позволяя им проводить больше времени с пациентами.

Кроме того, в большинстве стран мира системы здравоохранения не соответствуют системам в США и Европе. Развитие инструментов с открытым исходным кодом и доступных наборов данных предоставит правительствам и предпринимателям в остальном мире инструменты, необходимые им для обслуживания своих граждан.

Заключение

В этой главе вы узнали об истории применения методов машинного обучения в медицине. Мы начали с того, что дали обзор классических методов, таких как экспертные системы и байесовские сети, а затем переключились на более современные вопросы использования электронных медицинских карт и обработки результатов медицинских исследований. Завершает главу углубленное изучение конкретного примера – классификатора, предсказывающего прогрессирование диабетической ретинопатии. Мы также упомянули различные проблемы, с которыми сталкиваются системы глубокого обучения в здравоохранении. Мы вернемся к некоторым из этих проблем в нашей следующей главе, посвященной интерпретируемости систем глубокого обучения.

Глава 9

Генеративные модели

Все модели, которые мы рассмотрели до сих пор, занимаются в некотором роде трансляцией из входов в выходы. Вы создаете модель, которая принимает входные данные и производит выходные данные. Затем вы обучаете модель на входных выборках из набора данных, оптимизируя ее для получения наилучшей точности на выходе.

Генеративные модели (generative model) устроены иначе. Вместо того чтобы брать выборку в качестве входных данных, они производят выборку в качестве выходных данных. Вы можете обучить модель на библиотеке фотографий кошек, и она научится создавать новые изображения, похожие на кошек. Или, чтобы привести более уместный пример, вы могли бы обучить модель на библиотеке известных молекул лекарств, и она научилась бы генерировать новые «похожие на лекарства» молекулы для использования в качестве кандидатов на тестирование. Формально говоря, генеративная модель обучается на коллекции выборок, взятых из некоторого (возможно, неизвестного и наверняка очень сложного) распределения вероятностей. Работа генеративной модели состоит в том, чтобы производить новые выборки из того же распределения вероятностей.

В этой главе мы начнем с описания двух самых популярных типов генеративных моделей: *вариационных автоэнкодеров* (variational autoencoders, VAE) и *генеративных состязательных сетей* (generative adversarial networks, GAN). Затем мы обсудим несколько их применений в науках о жизни и проработаем несколько примеров кода.

Вариационные автоэнкодеры

Автоэнкодер – это модель, которая пытается сделать свои выходные данные равными своим входным данным. Вы тренируете модель на библиотеке выборок и настраиваете параметры модели так, чтобы на каждой выборке выход был как можно ближе к входу.

Это звучит тривиально. Разве автоэнкодер не может научиться передавать входные данные напрямую на выход без изменений? Если бы это было возможно, это действительно было бы тривиально, но автоэнкодеры обычно имеют архитектуры, которые делают сквозную передачу невозможной. Чаще всего это делается путем принудительной передачи данных через узкое место, как показано на рис. 9.1. Например, каждый из входов и выходов может содержать 1000 чисел, но между ними будет скрытый слой, содержащий только 10 чисел. Это заставляет модель учиться сжимать входные выборки. Она должна научиться представлять информацию из 1000 чисел, используя только 10 чисел.

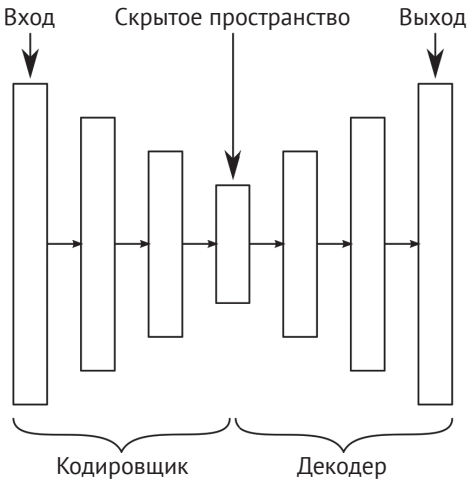


Рис. 9.1 ❖ Структура вариационного автоэнкодера

Если модель должна обрабатывать произвольные входные данные, это невозможно. Вы не можете выбросить 99 % информации, а затем восстановить входные данные! Но мы обрабатываем не произвольные входы, а только конкретные входы в обучающем наборе (и другие, похожие на них). Из всех возможных изображений намного менее 1 % выглядят как кошки. Автоэнкодеру нужно работать не со всеми возможными входными данными, а только с данными, полученными из определенного распределения вероятностей. Ему необходимо изучить «структуру» этого распределения, выяснить, как представить распределение с использованием гораздо меньшего количества информации, а затем научиться восстанавливать выборки на основе сжатой информации.

Теперь давайте разберем модель на рис. 9.1. Средний слой, который служит узким местом, называется *скрытым пространством* (latent space) автоэнкодера. Это пространство сжатых представлений выборок. Первая половина автоэнкодера называется *кодировщиком* (encoder). Его работа состоит в том, чтобы преобразовывать выборки в сжатые представления. Вторая половина называется *декодером* (decoder). Он получает сжатые представления из скрытого пространства и преобразует их обратно в исходные выборки.

Это дает нам первое представление о том, как автоэнкодеры можно использовать для генеративного моделирования. Итак, декодер берет векторы в скрытом пространстве и преобразует их в выборки. Давайте возьмем в скрытом пространстве случайный вектор и пропустим его через декодер. Просто выберем случайное значение для каждого компонента скрытого вектора. Если все идет хорошо, декодер должен создать совершенно новую выборку, которая все еще напоминает те выборки, на которых он был обучен.

Такой подход работает, но не очень хорошо. Проблема заключается в том, что кодер может создавать выходные векторы только на небольшой области скрытого пространства. Если вы выберете вектор где-нибудь за пределами этой области, то можете получить вывод, который не похож на обучающие выборки. Другими словами, декодер научился работать только для конкретных скрытых векторов, создаваемых кодером.

Вариационный автоэнкодер (VAE) реализует два способа решения этой проблемы. Во-первых, он добавляет составляющую функции потерь, заставляющую скрытые векторы следовать заданному распределению. Чаще всего они вынуждены иметь гауссово распределение со средним нулем и единичной дисперсией. Мы не позволяем кодировщику генерировать произвольные векторы. Наоборот, мы заставляем его генерировать векторы с известным распределением. Следовательно, если мы выберем случайные векторы из такого распределения, то можем рассчитывать на корректную работу декодера.

Во-вторых, во время обучения мы добавляем случайный шум к скрытому вектору. Кодер преобразует входную выборку в скрытый вектор, а затем мы случайным образом его немного меняем, перед тем как пропустить через декодер, и требуем, чтобы выходные данные были как можно ближе к исходной выборке. Это предотвращает чрезмерную чувствительность декодера к точным деталям скрытого вектора. Незначительное изменение скрытого вектора должно приводить лишь к незначительному изменению выхода.

Эти доработки значительно улучшили работу автоэнкодера, поэтому VAE являются популярным инструментом для генеративного моделирования. Во многих случаях они дают отличные результаты.

Генеративные состязательные сети

Генеративная состязательная сеть (GAN) имеет много общего с VAE. Она использует точно такую же сеть декодеров для преобразования скрытых векторов в выборки (за исключением того, что в GAN это называется *генератором*, а не декодером). Но процесс обучения устроен по-другому. Сеть обучается, передавая случайные векторы в генератор и непосредственно оценивая, насколько хорошо выходные данные соответствуют ожидаемому распределению. По сути вы создаете функцию потерь для измерения того, насколько хорошо сгенерированные выборки соответствуют обучающим выборкам, а затем используете эту функцию потерь для оптимизации модели.

Этот принцип работы выглядит простым и понятным лишь первые несколько секунд, пока вы не задумались о практической реализации. Не могли бы вы написать функцию потерь, чтобы измерить, насколько хорошо изображение напоминает кошку? Нет-нет, конечно нет! Вы не знаете, с чего начать. К счастью, GAN самостоятельно создает функцию потерь на основе данных.

GAN состоит из двух частей, как показано на рис. 9.2. Генератор берет случайные векторы и генерирует *синтетические выборки* (synthetic samples). Вторая часть, называемая *дискриминатором* (discriminator), пытается отличить сгенерированные выборки от реальных обучающих выборок. Дискриминатор берет выборку в качестве входных данных и выдает вероятность того, что это реальная обучающая выборка. Он действует как функция потерь для генератора.

Обе части обучаются одновременно. Случайные векторы поступают в генератор, а выход генератора – в дискриминатор. Параметры генератора настраиваются так, чтобы выход дискриминатора был как можно ближе к 1, тогда как параметры дискриминатора настроены так, чтобы его выход был как можно ближе к 0. Кроме того, дискриминатор настроен таким образом, чтобы при подаче на его вход реальных обучающих выборок выход дискриминатора был близок к 1.

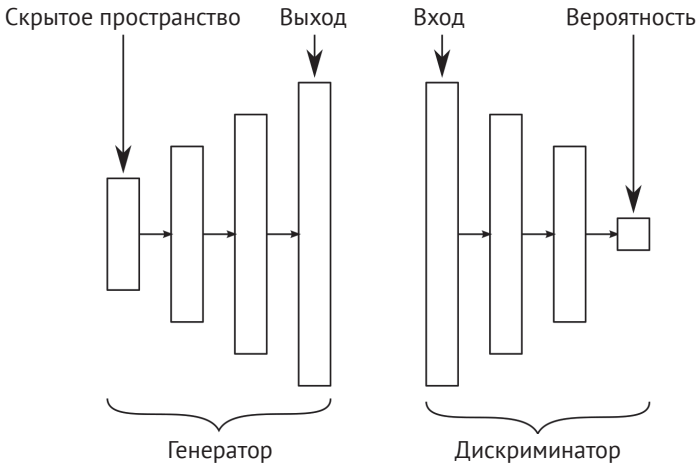


Рис. 9.2 ❖ Структура генеративной состязательной сети

Это и есть «состязательный» аспект. Вы можете думать об этом как о конкуренции между генератором и дискриминатором. Дискриминатор постоянно пытается как можно лучше отличить реальные выборки от поддельных. Генератор постоянно пытается все лучше обмануть дискриминатор.

Как и VAE, GAN являются популярным типом генеративной модели, которая дает хорошие результаты в различных приложениях. Эти два типа моделей имеют свои сильные и слабые стороны. Грубо говоря, можно сказать, что GAN склонны производить более качественные выборки, в то время как VAE склонны производить более качественные распределения. То есть отдельные выборки, сгенерированные GAN, будут больше похожи на обучающие выборки, тогда как *диапазон* выборок, сгенерированных VAE, будет более точно соответствовать диапазону обучающих выборок. Не воспринимайте это утверждение слишком буквально. Все зависит от конкретной проблемы и деталей модели. Кроме того, было предложено бесчисленное множество вариантов обоих подходов. Есть даже модели, которые сочетают VAE с GAN, чтобы взять от них лучшее. Это все еще очень активная область исследований, и очень часто публикуются новые идеи.

Применение генеративных моделей в науках о жизни

Теперь, когда мы познакомили вас с основами глубоких генеративных моделей, давайте поговорим о практическом применении. Вообще говоря, генеративные модели обладают новыми сверхспособностями. Во-первых, им присуща некоторая «креативность» – способность генерировать новые выборки на основе ранее изученного распределения. Это дает нам мощный инструмент для разработки новых лекарств или белков. Во-вторых, точное моделирование сложных систем с помощью генеративных моделей может пригодиться ученым при изучении сложных биологических процессов. Давайте обсудим эти идеи более подробно.

Генерация новых идей для соединений-прототипов

Основная часть усилий по открытию современных лекарств – это создание новых химических соединений. В настоящее время это в основном делается в на-

половину ручном режиме, когда опытные специалисты-химики предлагают модифицировать исходные структуры. Зачастую изображения молекул из текущего набора проецируют на экран, а сидящие в комнате опытные химики предлагают модификации структуры ядра молекулы. Некоторое подмножество предлагаемых молекул синтезируется и тестируется в лаборатории, и процесс повторяется до тех пор, пока не будет найдена подходящая молекула или программа не будет закрыта. Этот процесс имеет свои мощные преимущества, поскольку опирается на глубокую интуицию опытных химиков, способных выявить потенциальные недостатки структуры. Например, новая молекула чем-то напоминает им соединение, которое вызвало необъяснимую печеночную недостаточность у крыс.

В то же время участие человека не только дополняет, но и сильно ограничивает данный процесс. В мире не так много талантливых и опытных ведущих химиков, поэтому процесс разработки новых молекул не может выйти за пределы нескольких лабораторий. Кроме того, очень трудно организовать фармацевтическое производство в странах, не имеющих исторического опыта и вынужденных разворачивать исследования с нуля. Генеративные модели молекулярных структур могли бы помочь преодолеть эти ограничения. Если обучить модель определенному молекулярному представлению, она могла бы быстро предложить новые варианты молекул с подходящими свойствами. Такая модель способна выявить новые направления, упущенные специалистами-химиками. Стоит отметить, что такие алгоритмы разработки имеют и серьезные недостатки, как мы покажем далее в этой главе.

Разработка белков

Разработка новых энзимов и белков в наши дни является серьезным бизнесом. Искусственные энзимы широко используются в современном производстве. Наверняка некоторые энзимы содержатся в вашем стиральном порошке. Однако в целом разработка новых энзимов оказалась сложнее, чем ожидалось. В некоторых ранних работах по машинному обучению показано, что глубокие модели способны успешно прогнозировать свойства белка по его последовательности. И наоборот, генеративные модели нередко используются для поиска белковых последовательностей, имеющих заданные свойства.

Использование генеративных моделей для разработки белков может быть даже выгоднее, чем для разработки малых молекул. Эксперт-химик способен относительно точно предсказать свойства вещества с небольшой молекулой, но предсказание функций длинных белковых цепочек, с их конформациями и взаимозависимостями, выходит далеко за рамки возможностей человека. Зато с поиском скрытых закономерностей и зависимостей прекрасно справляются генеративные модели.

Инструменты для научного поиска

Генеративные модели – это перспективный инструмент научного поиска. Например, точная генеративная модель процесса развития тканей была бы чрезвычайно полезна для ученых-эмбриологов. Они могли бы проводить «синтетические эксперименты» для изучения развития тканей в различных условиях окружающей среды, используя генеративную модель для проведения быстрых симуляций. Это будущее пока далеко, так как нам нужны генеративные модели, сохраняющие

эффективность при изменении начальных условий. Разработчикам глубоких моделей еще предстоит проделать работу, выходящую за рамки возможностей современной техники. Тем не менее перспективы захватывают воображение, ведь генеративные модели позволят биологам точно моделировать физиологические процессы и развитие организма, чтобы проверять свои гипотезы.

Будущее генеративного моделирования

Генеративные модели сулят невероятные возможности! Первые GAN могли генерировать только размытые изображения, которые едва распознавались как лица. Последние GAN (на момент написания книги) способны генерировать изображения лиц, более или менее неотличимые от настоящих фотографий. Вполне вероятно, что в следующем десятилетии GAN смогут создавать генеративные видеоролики. Эти достижения имеют глубокие последствия для современного общества. В течение большей части прошлого столетия фотографии обычно использовались для доказательства преступлений, подтверждения личности и прочих официальных нужд. По мере развития генеративных инструментов этот формат доказательства будет дискредитирован, поскольку можно будет изготовить произвольную «фотографию», даже не имея навыков обработки изображений. Такое развитие событий станет серьезной проблемой для уголовного правосудия и даже для международных отношений.

В то же время появление высококачественного генеративного видео способно вызвать революцию в современной науке. Представьте себе качественные генеративные модели эмбрионального развития, способные смоделировать эффекты генетических модификаций CRISPR или детально воссоздать процесс развития эмбриона. Успехи генеративных моделей наверняка откликнутся и в других областях науки, например в метеорологии и фундаментальной физике, где особенно велика потребность в моделировании сложных систем. Однако стоит подчеркнуть, что эти улучшения пока остаются в будущем; чтобы добиться от таких моделей приемлемой стабильности, придется провести серьезные научные исследования.

Работа с генеративными моделями

Теперь давайте разберем пример кода. Мы будем учить VAE генерировать новые молекулы. Если точнее, он будет выводить строки SMILES. Такое представление имеет явные преимущества и недостатки по сравнению с другими представлениями, которые мы обсуждали ранее. С одной стороны, строки SMILES очень просты в работе. Каждая из них представляет собой последовательность символов, взятых из фиксированного алфавита. Это дает нам возможность использовать очень простую модель для их обработки. С другой стороны, строки SMILES должны соответствовать сложной грамматике. Если модель не принимает во внимание все тонкости грамматики, то большинство созданных строк окажется недействительными и не будет соответствовать какой-либо молекуле.

Первое, что нам нужно, – это набор строк SMILES для обучения модели. К счастью, MoleculeNet предоставляет нам широкий выбор. Для этого примера мы будем использовать набор данных MUV. Обучающий набор включает 74 469 молекул различных размеров и структур. Давайте начнем с загрузки:

```
import deepchem as dc
tasks, datasets, transformers = dc.molnet.load_muv()
train_dataset, valid_dataset, test_dataset = datasets
train_smiles = train_dataset.ids
```

Далее нам нужно определить «словарный запас», с которым будет работать наша модель. Каков список *токенов* (символов), которые могут появляться в строке? Какой длины могут быть строки? Мы можем определить это по обучающим данным, создав отсортированный список, состоящий из каждого символа, который появляется в любой обучающей молекуле:

```
tokens = set()
for s in train_smiles:
  tokens = tokens.union(set(s))
tokens = sorted(list(tokens))
max_length = max(len(s) for s in train_smiles)
```

Теперь нам нужно создать модель. Какую архитектуру мы должны использовать для кодера и декодера? Здесь нет готового ответа. Были опубликованы различные статьи, предлагающие разные модели. Мы воспользуемся классом DeepChem `AspuruGuzikAutoEncoder`, реализующим конкретную опубликованную модель. Он использует сверточную сеть для кодировщика и рекуррентную сеть для декодера. Вы не обязаны следовать примеру в выборе архитектуры, но если вас интересуют подробности, можете обратиться к оригинальной статье. Также обратите внимание, что в качестве функции скорости обучения мы используем `ExponentialDeCay`. Скорость изначально установлена на 0,001, затем немного снижается (умножается на 0,95) после каждой эпохи. Это способствует плавности оптимизации во многих задачах:

```
from deepchem.models.tensorgraph.optimizers import Adam, ExponentialDecay
from deepchem.models.tensorgraph.models.seqtoseq import AspuruGuzikAutoEncoder
model = AspuruGuzikAutoEncoder(tokens, max_length, model_dir='vae')
batches_per_epoch = len(train_smiles)/model.batch_size
learning_rate = ExponentialDecay(0.001, 0.95, batches_per_epoch)
model.set_optimizer(Adam(learning_rate=learning_rate))
```

Мы готовы обучать модель. Вместо использования стандартного метода `fit()`, принимающего объект `Dataset`, класс `AspuruGuzikAutoEncoder` предоставляет свой собственный метод `fit_sequence()`. Он принимает объект генератора Python, вырабатывающего последовательности токенов. В нашем случае это строки SMILES. Давайте запустим обучение в течение 50 эпох:

```
def generate_sequences(epochs):
  for i in range(epochs):
    for s in train_smiles:
      yield (s, s)

model.fit_sequences(generate_sequences(50))
```

Если все прошло хорошо, модель теперь сможет генерировать совершенно новые молекулы. Нам просто нужно выбрать случайные скрытые векторы и пропустить их через декодер. Давайте создадим партию из 1000 векторов длиной 196 (размер скрытого пространства модели).

Как отмечалось выше, не все выходные данные окажутся действительными строками SMILES. На самом деле действительными окажется лишь небольшая их часть. К счастью, мы можем использовать RDKit, чтобы с легкостью отфильтровать недействительные строки:

```
import numpy as np
from rdkit import Chem
predictions = model.predict_from_embeddings(np.random.normal(size=(1000,196)))
molecules = []
for p in predictions:
  smiles = ''.join(p)
  if Chem.MolFromSmiles(smiles) is not None:
    molecules.append(smiles)
for m in molecules:
  print(m)
```

Анализ вывода генеративной модели

Как упомянуто выше, некоторые выводы генеративной модели могут оказаться некорректными строками SMILES. Кроме того, многие из молекул, представленных строками SMILES, могут не соответствовать требованиям к молекулам лекарства. Таким образом, нам необходимо разработать правила, позволяющие быстро отфильтровать молекулы, которые не похожи на лекарства. Эти правила лучше всего объяснить на практическом примере. Давайте предположим, что это список SMILES, который поступил из нашей генеративной модели:

```
smiles_list = ['CCCCCCNNNCCOCC',
'O=C(O)C(=O)ON/C=N/CO',
'C/C=N/COCCNSCNCCNN',
'CCCNC(C(=O)O)c1cc(OC(OC)[SH](=O)=O)ccc1N',
'CC1=C2C=CCC(=CC(Br)=CC=C1)C2',
'CCN=NNNC(C)OOCOOOOOCOOO',
'N#CNCCCCCOCCOC1COCNN1CCCCCCCCCCCCCCCCCCCCOOOOOSNNCCCCCSCSCCCCCCCCCCOCOOOSS',
'CCCC(=O)NC1=C(N)C=COO1',
'CCCSc1cc2nc(C)cnn2c1NC',
'CONCN1N=NN=CC=C1CC1SSS1',
'CCCOc1ccccc1OSNNOCCNCSNCCN',
'C[SH]1CCCN2CCN2C=C1N',
'CC1=C(C#N)N1NCCC1=COOO1',
'CN(NCNNNN)C(=O)CCSCc1ccco1',
'CCCN1CCC1CC=CC1=CC=S1CC=O',
'C/N=C/c1ccccc1',
'Nc1cccooo1',
'CCOc1ccccc1CCCNC(C)c1nccs1',
'CNNNNNNc1nocc1CCNNC(C)C',
'COC1=C(CON)C=C2C1=C(C)c1ccccc12',
'CCOCCCCNN(C)C',
'CCCN1C(=O)CNC1C',
'CCN',
'NCCNCc1cccc2c1C=CC=CC=C2',
'CCCCCN(NNNCNCCCCCCCCCCSCCCCCCCCCCCCCCCNCCNCCCCSSCSSSSSSSCCCCCCCCCCCCCSCCCCCSC)C(O)OCCN',
'CCCS1=CC=C(C)N(CN)C2NCC2=C1',
```

```
'CCNCCCCCCOc1cccc(F)c1',
'NN1O[SH](CCCCO)C12C=C2',
'Cc1cc2cccc3c(CO)cc-3ccc-2c1']
```

Первым шагом в нашем анализе будет изучение молекул и выявление кандидатов на удаление из списка. Для исследования молекул мы воспользуемся средствами RDKit, входящего в состав DeepChem. Чтобы оценить строки SMILES, мы должны сначала преобразовать их в объекты молекул. Мы можем сделать это, используя следующий список соответствия:

```
molecules = [Chem.MolFromSmiles(x) for x in smiles_list]
```

Один из факторов, который мы можем рассмотреть, – это размер молекул. Молекулы менее чем из 10 атомов вряд ли будут генерировать энергию взаимодействия, достаточную для получения измеримого отклика в биологическом опыте. Большие молекулы, содержащие более 50 атомов, обычно плохо растворяются в воде и создают другие проблемы в биологических опытах. Мы можем получить приблизительную оценку размера молекулы, рассчитав в каждой молекуле количество атомов, не являющихся атомами водорода. Для удобства мы отсортируем массив так, чтобы нам было легче понять распределение. Если бы у нас был больший список молекул, мы, вероятно, захотели бы сгенерировать гистограмму для этого распределения. Следующая строка кода создает список количества атомов в каждой молекуле:

```
print(sorted([x.GetNumAtoms() for x in molecules]))
```

и выводит следующий результат:

```
[3, 8, 9, 10, 11, 11, 12, 12, 13, 14, 14, 14, 15,
16, 16, 16, 17, 17, 17, 17, 18, 19, 19, 20, 20, 22, 24, 69, 80]
```

Мы можем видеть, что есть четыре очень маленькие и две большие молекулы. Мы можем использовать другое выражение списка, чтобы удалить молекулы с менее чем 10 или более чем 50 атомами:

```
good_mol_list = [x for x in molecules if x.GetNumAtoms() > 10 and x.GetNumAtoms() < 25]
print(len(good_mol_list))
```

```
23
```

Приведенное выше выражение уменьшило наш предыдущий список из 29 молекул до 23.

На практике для оценки качества генерируемых молекул мы можем использовать ряд других вычисляемых свойств. В некоторых недавних публикациях о генеративных моделях для определения пригодности молекулы используют *вычисляемые молекулярные свойства*. Один из наиболее распространенных методов определения схожести молекул с известными лекарственными средствами известен как *количественная оценка сходства лекарств* (quantitative estimate of druglikeness, QED). Методика QED, предложенная Бикертоном (Bickerton) и его коллегами[1], сравнивает распределение набора свойств, рассчитанных для моле-

[1] *Bickerton Richard G. et al*. Quantifying the Chemical Beauty of Drugs. 2012 // **http://dx.doi.org/10.1038/nchem.1243**.

кулы, с распределениями тех же свойств в продаваемых лекарствах. Показатель совпадения варьируется от 0 до 1,0, при этом молекулы с показателем около 1,0 считаются наиболее похожими на лекарства. В качестве оценки мы рассчитаем QED для оставшихся молекул и отбросим те, у которых показатели меньше 0,5.

Мы рассчитаем значения QED для оставшихся молекул при помощи RDKit и сохраним только молекулы с QED > 0,5:

```
qed_list = [QED.qed(x) for x in good_mol_list]
final_mol_list = [(a,b) for a,b in zip(good_mol_list,qed_list) if b > 0.5]
```

В качестве нашего последнего шага мы можем визуализировать химические структуры `final_mol_list` и соответствующие оценки QED:

```
MolsToGridImage([x[0] for x in final_mol_list],
                molsPerRow=3,useSVG=True,
                subImgSize=(250, 250),
                legends=[f"{x[1]:.2f}" for x in final_mol_list])
```

Результат показан на рис. 9.3.

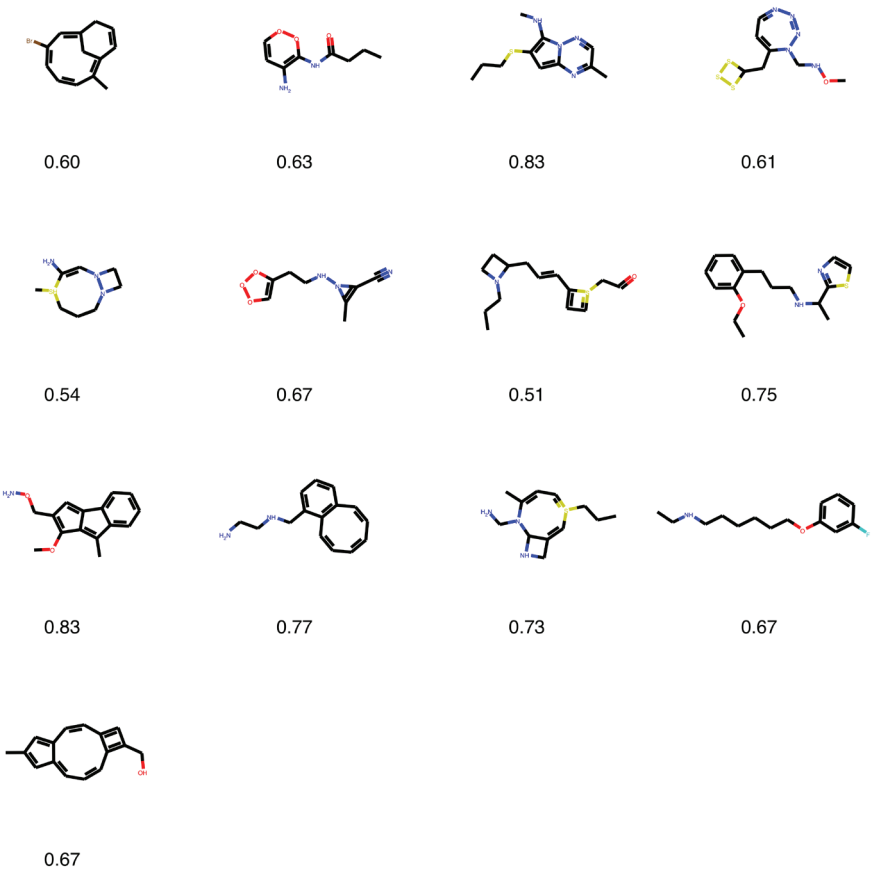


Рис. 9.3 ❖ Химическая структура генерируемых молекул вместе с их показателями QED

Хотя эти структуры являются действительными и имеют достаточно высокие оценки QED, они все же могут быть химически нестабильными. Стратегии выявления и удаления проблемных молекул обсуждаются в следующей главе.

Заключение

Хотя генеративные модели доказали свою полезность в качестве инструмента для создания новых молекул, для массового применения таких моделей необходимо решить некоторые ключевые проблемы. Прежде всего надо обеспечить, чтобы сгенерированные молекулы были химически стабильными и вообще могли быть физически синтезированы. Один из современных методов оценки качества молекул на выходе генеративной модели заключается в том, чтобы следить за соблюдением стандартных правил химической валентности, то есть гарантировать, что каждый атом углерода имеет четыре связи, каждый атом кислорода имеет две связи, каждый атом фтора имеет одну связь и т. д. Эти факторы становятся особенно важными при декодировании из скрытого пространства в представлении SMILES. Хотя генеративная модель, возможно, и выучила грамматику SMILES, некоторые нюансы могут быть упущены.

Тот факт, что молекула подчиняется стандартным правилам валентности, не обязательно гарантирует ее химическую стабильность. В некоторых случаях генеративная модель может предлагать молекулы, содержащие неустойчивые функциональные группы. В качестве примера рассмотрим молекулу на рис. 9.4. Функциональная группа, выделенная в кружке, известная как *полуацеталь*, легко разлагается.

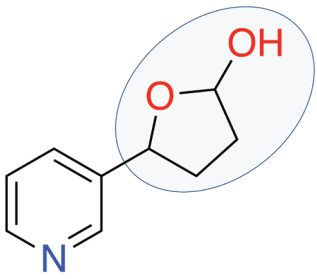


Рис. 9.4 ❖ Молекула, содержащая нестабильную группу

На практике вероятность того, что генеративная молекула окажется физически реализуемой и химически стабильной, очень мала. Существуют десятки химических функциональных групп, которые, как известно, являются нестабильными или чрезмерно реакционно-активными. Профессиональные химики знают, как избежать введения таких функциональных групп при синтезе лекарственных молекул. Необходимо каким-то образом передать эти специфические знания генеративной модели. Одним из способов является составление набора фильтров, позволяющих отбрасывать проблематичные молекулы. В главе 11 мы обсудим некоторые из этих фильтров и их применение в виртуальном скрининге. Для оценки виртуальных молекул, созданных генеративной моделью, можно применить

методы, разработанные для выявления потенциально проблематичных соединений в скрининге.

Чтобы проверить на деле биологическую активность молекулы, предложенной генеративной моделью, эту молекулу сначала должен синтезировать химик. Наука органического химического синтеза имеет богатую историю, насчитывающую более 100 лет. За это время химики разработали тысячи химических реакций, пригодных для синтеза лекарств. Для синтеза молекулы, подобной обычному лекарственному средству, обычно требуется от 5 до 10 химических реакций, часто называемых стадиями. Простые молекулы синтезируются относительно легко, но для синтеза некоторых сложных молекул может потребоваться более 20 стадий. Хотя попытки автоматизировать планирование органического синтеза длятся более 50 лет, большая часть синтезов все еще управляется человеческой интуицией в сопровождении метода проб и ошибок.

К счастью, благодаря последним разработкам в области глубокого обучения появились новые способы планирования синтеза лекарственных молекул. Несколько научных групп опубликовали работы, посвященные использованию глубокого обучения для планирования стадий синтеза. В качестве входных данных модели дается молекула, часто называемая продуктом, и набор стадий, которые использовались для синтеза этой молекулы. Обучаясь на десятках тысяч молекул и соответствующих стадиях синтеза, глубокая нейронная сеть способна уловить взаимосвязь между молекулами продукта и стадиями синтеза. Когда представлена новая молекула, модель предлагает набор реакций, потенциально подходящих для синтеза молекулы. Специалистам-химикам были представлены для оценки планы синтеза, предложенные генеративными моделями. Эксперты сделали вывод, что планы генеративных моделей сопоставимы по качеству с планами опытных химиков.

Применение глубокого обучения в органическом синтезе является относительно новой областью. Есть надежда, что эта область продолжит развиваться и станет важным инструментом для химиков-органиков. Можно представить себе не столь отдаленное будущее, когда машинное планирование синтеза будет объединено с роботизированным лабораторным оборудованием и образует полностью автоматическую платформу синтеза органических веществ. Но на этом пути придется преодолеть еще много трудностей.

Одним из потенциальных препятствий на пути широкого внедрения глубокого обучения в органический синтез является *доступность данных*. Большая часть информации, подходящей для обучения моделей, хранится в базах данных, принадлежащих ограниченному кругу организаций. Если эти организации решат использовать данные только для своих внутренних исследований, на поле останется очень мало игроков.

Другим фактором, сдерживающим развитие генеративных моделей, является качество прогностических моделей, применяемых для управления генерацией молекул. Независимо от архитектуры генеративной модели нам нужны инструменты для оценки генерируемых молекул и прогнозирования направлений поиска новых молекул. В некоторых случаях удается разработать надежные прогностические модели. В других случаях модели получаются менее надежными. Генеративные модели можно тестировать на внешних оценочных наборах, но область действия прогностической модели иногда определяется с трудом. Эта об-

ласть, также известная как *область применимости* (domain of applicability), отражает способность модели к экстраполяции навыков на незнакомые молекулы. Область применимости не поддается четкому определению, поэтому иногда трудно определить, насколько хорошо прогностическая модель будет работать с новыми молекулами, созданными генеративной моделью.

Генеративные модели являются относительно новым методом, и будет интересно наблюдать за развитием данной научной области в ближайшие годы. Отдача от применения генеративных моделей будет расти по мере развития методов планирования синтеза и улучшения прогностических моделей.

Глава 10

Интерпретация глубоких моделей

На данный момент мы рассмотрели много примеров обучения глубоких моделей, решающих разные задачи. В каждом случае мы собираем какие-то данные, строим модель и обучаем ее до тех пор, пока она не даст правильные результаты на проверочных и оценочных данных. Затем мы похлопываем друг друга по спине, объявляем, что проблема решена, и переходим к следующей проблеме. В конце концов, у нас есть модель, которая дает правильные прогнозы для входных данных. Что еще мы можем хотеть?

Но зачастую это лишь начало! Закончив обучение модели, вы можете задать много важных вопросов. Как работает модель? Какие аспекты входной выборки привели к определенному прогнозу? Можете ли вы доверять прогнозам модели? Насколько они точны? Есть ли ситуации, когда модель может потерпеть неудачу? *Что именно* она «выучила»? И может ли это привести к новому пониманию данных, на которых происходило обучение?

Все эти вопросы относятся к теме *интерпретируемости*. Она охватывает все, что вы можете захотеть от модели, помимо механического использования ее для прогнозирования. Это очень обширный предмет, и методы, которые он охватывает, столь же разнообразны, как и вопросы, на которые он пытается ответить. Мы не надеемся раскрыть всю тему в одной главе, но попытаемся, по крайней мере, обрисовать в общих чертах некоторые наиболее важные подходы.

Для этого мы вернемся к примерам из предыдущих глав. Раньше мы просто обучали модели делать предсказания, проверяли их точность и затем считали нашу работу завершенной. Теперь мы копнем глубже и посмотрим, чему еще мы можем научиться.

Как объяснить предсказания?

Предположим, вы обучили модель распознавать фотографии различных видов транспортных средств. Вы запускаете модель на проверочном наборе и обнаруживаете, что она точно различает автомобили, лодки, поезда и самолеты. Означает ли это, что модель готова к запуску в производство? Уверены ли вы, что модель продолжит давать точные ответы в будущем?

Может быть и так, но если неправильные ответы приводят к серьезным последствиям, не помешает дополнительная проверка. Было бы полезно понимать,

почему модель дает свои конкретные прогнозы. Она действительно смотрит на транспортное средство или цепляется за посторонние аспекты изображения? Фотографии автомобилей обычно также содержат изображения дороги. Самолеты, как правило, изображены на фоне неба. Фотографии поездов обычно содержат рельсы, а лодки сфотографированы на фоне воды. Если модель на самом деле определяет фон, а не транспортное средство, она может хорошо работать на тестовом наборе, но плохо в непредвиденных случаях. Лодка на фоне неба может быть классифицирована как самолет, а автомобиль, проезжающий мимо воды, может быть распознан как лодка.

Другая потенциальная проблема может заключаться в чрезмерной фиксации модели на мелких деталях. Возможно, модель выделяет не изображения *автомобилей*, а изображения, содержащие *номерные знаки*. Или, вероятно, она очень хорошо выявляет спасательные круги и научилась ассоциировать их с изображениями лодок. В большинстве случаев это хорошо срабатывает, но абсолютно не работает, когда на фото изображен автомобиль, проезжающий мимо бассейна, и на заднем плане виден спасательный круг.

Возможность достоверно объяснить, *почему* модель сделала прогноз, является важной частью интерпретируемости. Когда модель выделяет фотографию автомобиля, хочется убедиться, что она обучилась узнавать именно автомобиль, а не дорогу или одну маленькую характерную часть автомобиля. Короче говоря, вы хотите знать, что модель дала правильный ответ *по правильным причинам*. Это дает вам уверенность, что модель и в будущем сработает правильно на новых данных.

В качестве конкретного примера давайте вернемся к модели диабетической ретинопатии из главы 8. Напомним, что эта модель принимает изображение сетчатки в качестве входного сигнала и прогнозирует наличие и степень диабетической ретинопатии у пациента. Между входом и выходом находятся десятки слоев и более восьми миллионов обученных параметров. Мы хотим понять, почему определенный ввод привел к определенному результату, но мы не можем надеяться узнать это, просто взглянув на модель. Ее сложность далеко за пределами человеческого понимания.

Для ответа на этот вопрос разработано много методов. Мы применим один из самых простых из них, называемый *картированием значимости*[1] (saliency mapping). Суть этого метода заключается в том, чтобы спросить, какие пиксели входного изображения наиболее важны (или «значимы») для определения выходных данных. В некотором смысле, конечно, каждый пиксель важен. Выход является чрезвычайно сложной нелинейной функцией всех входов. На произвольном изображении любой пиксель может содержать признаки заболевания. Но в конкретном изображении это делает только часть из них, и мы хотим знать, какие именно пиксели задействованы.

При построении карт значимости используется простая аппроксимация: просто берется производная от выходов по всем входам. Если регион изображения не содержит признаков заболевания, небольшие изменения любого отдельного пикселя в этом регионе должны слабо влиять на вывод. Следовательно, производная

[1] *Simonyan K., Vedaldi A., and Zisserman A.* Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. 2014 // **https://arxiv.org/abs/1312.6034**.

должна быть маленькой. Положительный диагноз включает в себя корреляции между многими пикселями. Когда эти корреляции отсутствуют, они не могут быть созданы только за счет изменения одного пикселя. Но когда они присутствуют, изменение любого из коррелирующих пикселей может потенциально усилить или ослабить результат. Производная должна быть наибольшей в «важных» регионах, на которые обращает внимание модель.

Давайте поработаем с кодом. Сначала нам нужно построить модель и перезагрузить обученные значения параметров:

```
import deepchem as dc
import numpy as np
from model import DRModel
from data import load_images_DR

train, valid, test = load_images_DR(split='random', seed=123)
model = DRModel(n_init_kernel=32, augment=False, model_dir='test_model')
model.restore()
```

Теперь мы можем использовать модель, чтобы делать прогнозы по выборкам. Например, давайте проверим прогнозы для первых десяти тестовых выборок:

```
X = test.X
y = test.y
for i in range(10):
  prediction = np.argmax(model.predict_on_batch([X[i]]))
  print('True class: %d, Predicted class: %d' % (y[i], prediction))
```

Так выглядит вывод:

```
True class: 0, Predicted class: 0
True class: 2, Predicted class: 2
True class: 0, Predicted class: 0
True class: 0, Predicted class: 0
True class: 3, Predicted class: 0
True class: 2, Predicted class: 2
True class: 0, Predicted class: 0
True class: 0, Predicted class: 0
True class: 0, Predicted class: 0
True class: 2, Predicted class: 2
```

Модель правильно выдает 9 прогнозов из первых 10 выборок, что неплохо. Но на что она смотрит, когда делает свои прогнозы? Карта значимости может дать нам ответ, а DeepChem легко сформирует такую карту:

```
saliency = model.compute_saliency(X[0])
```

Метод `compute_saliency()` принимает входной массив для конкретной выборки и возвращает производную каждого вывода по каждому входу. Мы можем лучше понять, что это значит, взглянув на форму результата:

```
print(saliency.shape)
```

Вывод на печать сообщает, что это массив формы (5, 512, 512, 3). `X[0]` – это 0-е входное изображение, которое представляет собой массив формы (512, 512, 3),

причем последним измерением являются три цветовых компонента. Кроме того, модель имеет пять выходов – вероятностей принадлежности выборки к каждому из пяти классов. Значимость содержит производную каждого из пяти выходов по отношению к каждому из 512×512×3 входов.

Эти данные нужно немного обработать, чтобы сделать их более полезными. Во-первых, нас интересует абсолютное значение каждого элемента. Не важно, должен ли пиксель быть темнее или светлее, чтобы увеличить выходной сигнал, мы хотим лишь знать, влияет ли он на выход – да или нет. Затем мы хотим сократить данные до одного числа на пиксель. Это можно сделать разными способами, но сейчас мы просто пройдемся по всем измерениям. Если какой-либо компонент цвета влияет на любое из выходных предсказаний, это делает пиксель важным. Наконец, мы нормализуем значения в диапазоне от 0 до 1:

```
sal_map = np.sum(np.abs(saliency), axis=(0,3))
sal_map -= np.min(sal_map)
sal_map /= np.max(sal_map)
```

Давайте посмотрим, как это выглядит. На рис. 10.1 представлено изображение, на котором модель правильно идентифицирует признаки тяжелой диабетической ретинопатии. Исходное изображение слева, а справа белым цветом выделены значащие области.

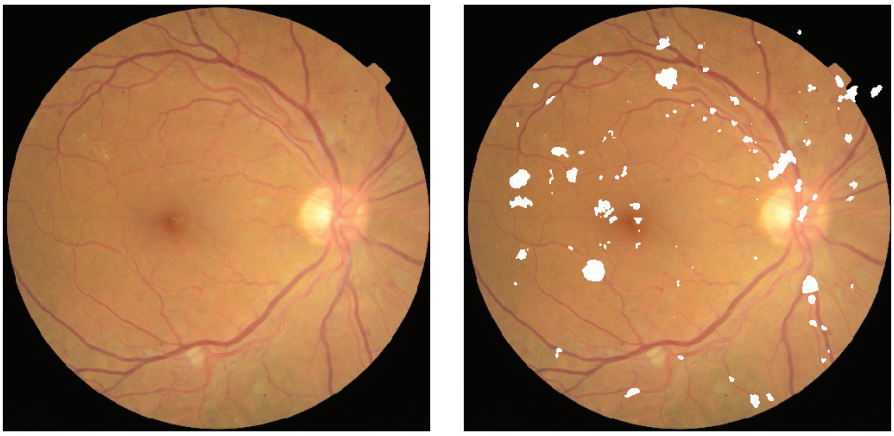


Рис. 10.1 ❖ Карта значимости для изображения глазного дна с тяжелой диабетической ретинопатией

Прежде всего мы замечаем, что значимость широко распространена по всей сетчатке. Однако распределение пятен явно неоднородное. Значимость сосредоточена вдоль кровеносных сосудов, особенно в местах, где сосуды разветвляются. Действительно, некоторые из признаков, которые врач ищет для диагностики диабетической ретинопатии, включают аномальные кровеносные сосуды, кровотечения и рост новых кровеносных сосудов. Похоже, что модель сосредоточила свое внимание на правильных частях изображения – на тех же самых, которые наиболее внимательно рассматривал бы доктор.

Оптимизация входов

Картирование значимости и аналогичные методы говорят нам, на какую информацию опиралась модель, когда делала прогноз. Но как именно она интерпретировала эту информацию? Модель диабетической ретинопатии ориентирована на кровеносные сосуды, но *что* она ищет, чтобы отличить здоровые кровеносные сосуды от больных? Точно так же, когда модель ищет фотографию лодки, полезно знать, что она сделала вывод на основе пикселей, составляющих лодку, а не тех, которые составляют фон. Но что именно в этих пикселях заставило модель сделать вывод, что она видит лодку? Вывод был основан на цвете? На форме? На сочетании мелких деталей? Могут ли существовать обманчивые изображения, которые модель одинаково уверенно (но неверно) идентифицирует как лодку? Как именно модель «думает» о внешнем виде лодки?

Обычный подход к ответу на эти вопросы заключается в поиске входных данных, которые максимизируют вероятность прогнозирования. Если рассматривать все возможные варианты данных на входе, какие из них приведут к появлению на выходе модели прогноза с максимальной вероятностью? Изучив эти входные данные, вы можете увидеть, что модель «ищет» на самом деле. Иногда оказывается, что видение модели, мягко говоря, сильно отличается от того, что вы ожидаете! На рис. 10.2 показаны изображения, которые были оптимизированы для получения четких прогнозов при подаче в *высококачественную* модель распознавания изображений. Модель относит каждое изображение к соответствующей категории с очень высокой степенью достоверности, но для человека они почти не имеют сходства!

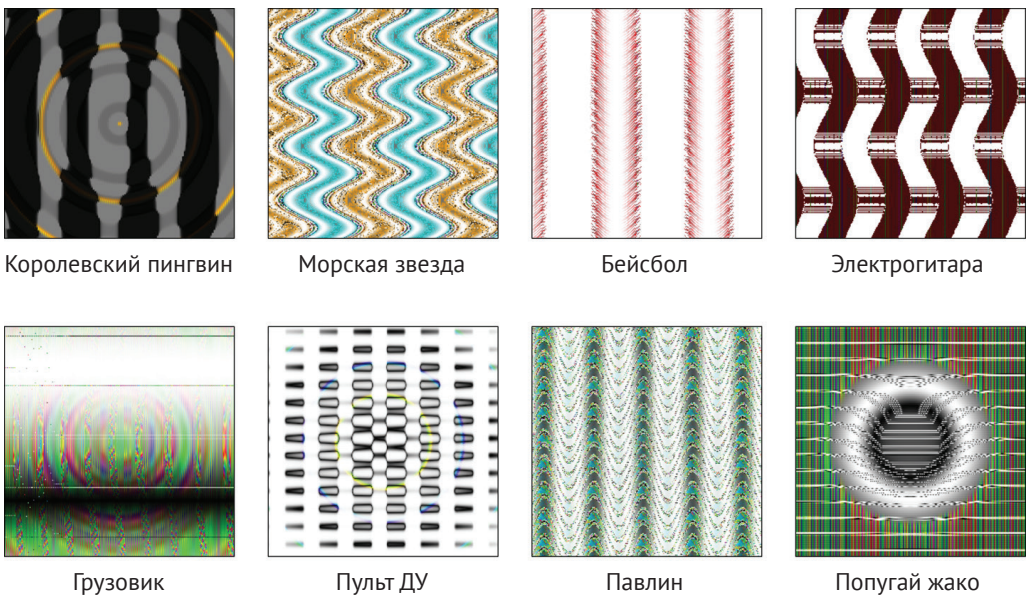


Рис. 10.2 ❖ Изображения, которые обманывают высококачественную модель распознавания изображений.
Источник: **https://arxiv.org/abs/1412.1897**

В качестве примера рассмотрим модель связывания факторов транскрипции из главы 6. Напомним, что эта модель принимает последовательность ДНК в качестве входных данных и прогнозирует, содержит ли последовательность сайт связывания для фактора транскрипции JUND. Как для модели выглядит сайт связывания? Нам хотелось бы рассмотреть все возможные последовательности ДНК и найти те, для которых модель наиболее уверенно предсказывает наличие сайта связывания.

К сожалению, мы не можем действительно рассмотреть все возможные входные данные. Существует 4101 возможная последовательность ДНК длиной 101. Если мы будем тратить одну наносекунду на изучение каждой последовательности, то для перебора всех сочетаний потребуется в несколько раз больше времени, чем возраст Вселенной. Вместо этого нам нужна стратегия ограничения количества входных данных.

Один из подходов – просто посмотреть на последовательности в обучающем наборе. В данном случае это действительно разумная стратегия. Обучающий набор охватывает десятки миллионов оснований из реальных хромосом, поэтому он, вероятно, является хорошим представлением входных данных, с которыми модель столкнется на практике. На рис. 10.3 приведено 10 последовательностей из обучающего набора, для которых модель дает наивысший выход. Предполагается, что каждая из них имеет сайт связывания с вероятностью более 97 %. Девять из них действительно имеют сайты связывания, в то время как одна является ложноположительной. Чтобы определить, на что обращает внимание модель, мы составили карту значимости для каждой последовательности и раскрасили основания в соответствии с их значимостью.

Рис. 10.3 ❖ 10 обучающих примеров с самыми высокими прогнозируемыми результатами. Галочками помечены выборки, которые содержат действительные сайты связывания

Глядя на эти входные данные, мы можем сразу увидеть базовый шаблон, который распознает модель: TGA-*xx*-TCA, где *xx* состоит из одного или двух оснований, обычно C или G. Значимость указывает, что модель также обращает некоторое внимание на два основания с каждой стороны шаблона. Предшествующее основание может быть A, C или G, а последующее основание – всегда C или T. Это согласуется с известным мотивом связывания для JUND, который представлен на рис. 10.4 в виде позиционно-весовой матрицы.

Рис. 10.4 ❖ Известный мотив связывания для JUND, представленный в виде позиционно-весовой матрицы. Высота каждой буквы указывает на вероятность появления этого основания в соответствующей позиции

Одна последовательность, ошибочно предсказанная как имеющая сайт связывания, не содержит этот шаблон. Вместо этого он имеет несколько близко расположенных друг к другу повторений шаблона TGAC. Это похоже на начало истинного мотива связывания, но за ним никогда не следует шаблон TCA. Очевидно, наша модель научилась определять истинный мотив связывания, но может впасть в заблуждение, когда встречает несколько следующих друг за другом неполных шаблонов.

Обучающие выборки не всегда бывают хорошим представлением полного диапазона возможных входных данных. Если ваш тренировочный набор целиком состоит из фотографий транспортных средств, он ничего не говорит вам о том, как модель отреагирует на другие входные данные. Возможно, если показать модели фотографию снежинки, она уверенно опознает ее как лодку. Можно предположить, что существуют входные данные, которые не имеют ничего общего с фотографиями (может быть, просто геометрические узоры или даже случайный шум), которые модель опознает как лодки. Чтобы проверить эту возможность, мы не можем полагаться на уже имеющиеся у нас данные. Вместо этого нам нужно позволить модели рассказать нам, что она ищет. Мы начнем с абсолютно случайного ввода, а затем используем алгоритм оптимизации для изменения ввода таким образом, чтобы увеличить точность модели.

Давайте попробуем сделать это для модели связывания факторов транскрипции. Мы начнем с генерации совершенно случайной последовательности и вычисления прогноза модели для нее:

```
best_sequence = np.random.randint(4, size=101)
best_score = float(model.predict_on_batch([dc.metrics.to_one_hot(best_sequence, 4)]))
```

Теперь оптимизируем ввод. Мы случайным образом выбираем позицию в последовательности и вписываем туда новое основание. Если это изменение приводит к увеличению выхода, мы сохраняем его. В противном случае отменяем изменение и пробуем что-то еще:

```
for step in range(1000):
  index = np.random.randint(101)
  base = np.random.randint(4)
  if best_sequence[index] != base:
    sequence = best_sequence.copy()
    sequence[index] = base
    score = float(model.predict_on_batch([dc.metrics.to_one_hot(sequence, 4)]))
    if score > best_score:
      best_sequence = sequence
      best_score = score
```

Этот процесс быстро приводит нас к последовательностям, которые максимизируют вероятность на входе. Обычно в течение 1000 шагов мы достигаем вероятности 1,0.

На рис. 10.5 показано 10 последовательностей, сгенерированных процессом оптимизации. Выделены все экземпляры трех наиболее распространенных шаблонов связывания (TGACTCA, TGAGTCA и TGACGTCA). Каждая последовательность содержит, по крайней мере, одно вхождение одного из этих шаблонов, но обычно мы видим три или четыре вхождения. Итак, последовательности, которые макси-

мизируют вывод модели, имеют именно те свойства, которые мы ожидаем от них, что дает нам уверенность в правильности работы модели.

Рис. 10.5 ❖ Примеры последовательностей, которые были оптимизированы для достижения максимальной точности модели

Прогнозирование неопределенности

Даже если вы убедились в том, что модель дает точные прогнозы, все равно остается важный вопрос: *насколько* они точны? В науке нас редко устраивает просто число; мы хотим знать *неопределенность* (uncertainty) каждого прогноза. Если модель выдает 1,352, означает ли это, что истинное значение находится между 1,351 и 1,353? Или оно лежит где-то между 0 и 3?

В качестве конкретного примера мы воспользуемся моделью растворимости из главы 4. Напомним, что эта модель принимает в качестве входных данных молекулу, представленную в виде молекулярного графа, и выводит число, указывающее, насколько легко молекула растворяется в воде. Мы построили и обучили модель с помощью следующего кода:

```
tasks, datasets, transformers = dc.molnet.load_delaney(featurizer='GraphConv')
train_dataset, valid_dataset, test_dataset = datasets
model = GraphConvModel(n_tasks=1, mode='regression', dropout=0.2)
model.fit(train_dataset, nb_epoch=100)
```

Когда мы впервые исследовали эту модель, мы оценили ее точность на тестовом наборе и заявили, что удовлетворены. Теперь мы можем попробовать еще лучше оценить ее точность.

Очень простая вещь, которую мы могли бы попытаться сделать, – это просто вычислить *среднеквадратичную ошибку* (root-mean-squared, RMS) прогнозов модели на тестовом наборе:

```
y_pred = model.predict(test_dataset)
print(np.sqrt(np.mean((test_dataset.y-y_pred)**2)))
```

Расчет выдает среднеквадратическую ошибку 0,396. Должны ли мы использовать это значение как меру неопределенности для всех прогнозов, сделанных моделью? *Если* набор тестов представляет все входные данные, на которых будет использоваться модель, и *если* все ошибки следуют одному распределению, это может быть разумным решением. К сожалению, ни одно из этих предположений не является безопасным! Ошибка у одних предсказаний больше, чем у других, и в зависимости от конкретных молекул, попадающих в проверочный набор, их средняя ошибка может быть либо выше, либо ниже, чем на практике.

Нам хотелось бы связать с каждым результатом конкретную неопределенность. Мы хотим заранее знать, какие прогнозы являются более точными, а какие менее точными. Чтобы добиться этого, нам нужно более тщательно рассмотреть множество факторов, влияющих на ошибки в предсказаниях модели[1]. Как вы увидите дальше, следует рассматривать два принципиально разных типа неопределенности.

Рисунок 10.6 показывает истинную и предсказанную растворимость молекул на обучающем наборе. Модель воспроизводит обучающий набор очень хорошо, но все же не идеально. Точки распределены в диагональной полосе с конечной шириной. Несмотря на то что модель проверяли на тех же образцах, на которых она обучалась, на выходе все равно остается некоторая неопределенность. Следует ожидать, что модель будет демонстрировать *как минимум* такую же неопределенность на незнакомых данных.

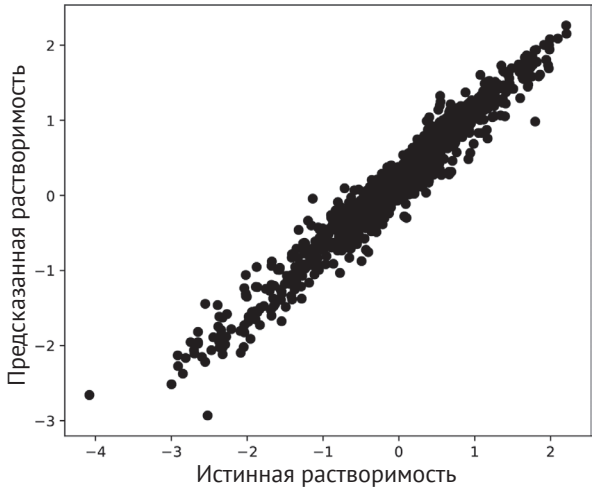


Рис. 10.6 ❖ Истинные и предсказанные растворимости для молекул в тренировочном наборе

Обратите внимание, что мы использовали только обучающий набор. Неопределенность может быть полностью вычислена на основе информации, доступной во время обучения. Это означает, что мы можем обучить модель предсказанию неопределенности! Мы можем добавить в модель еще один набор результатов: для каждого значения, которое она прогнозирует, она также выведет оценку неопределенности этого прогноза.

Теперь посмотрите на рис. 10.7. Мы повторили процесс обучения 10 раз, получив 10 разных моделей. Затем мы воспользовались каждой из них, чтобы предсказать растворимость 10 молекул из проверочного набора. Все модели были обучены на одних и тех же данных, и они демонстрируют схожие ошибки в обучающем наборе, но дают разные предсказания для молекул проверочного набора! Для

[1] *Kendall A., and Gal Y.* What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? 2017 // **https://arxiv.org/abs/1703.04977**.

каждой молекулы мы получили различные значения растворимости в зависимости от используемой модели.

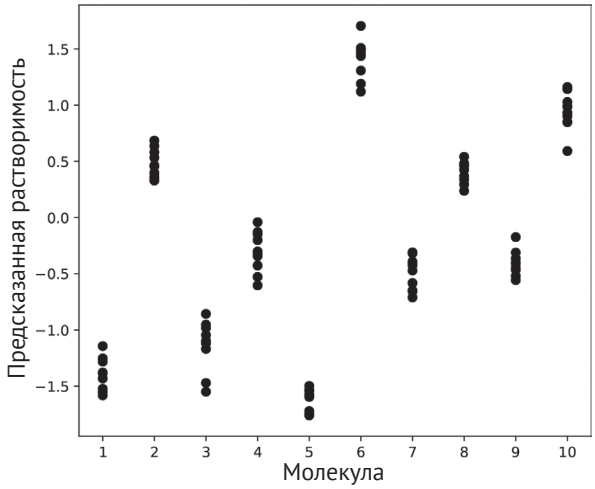


Рис. 10.7 ❖ Растворимость 10 молекул из проверочного набора, предсказанная группой моделей, обученных на одних и тех же данных

Это принципиально иной тип неопределенности, известный как *эпистемическая неопределенность* (epistemic uncertainty). Она возникает из-за того, что многие различные модели в равной степени соответствуют данным обучения, и мы не знаем, какая из них «лучшая».

Простой способ измерить эпистемическую неопределенность состоит в том, чтобы обучить большое количество моделей и сравнить их результаты, как мы сделали на рис. 10.7. Однако зачастую это слишком дорого. Если у вас есть большая и сложная модель, на обучение которой уходят недели, вы не сможете повторять этот процесс много раз.

Гораздо более быстрой альтернативой является обучение одной модели с использованием исключения, а затем многократное прогнозирование каждого выхода с разными масками исключения. Обычно исключение производится только во время тренировки. Если на каждом этапе обучения 50 % выходных данных слоя случайным образом устанавливают на 0, то во время теста вы вместо этого умножаете каждый выходной сигнал на 0,5. Но сейчас мы не будем этого делать. Давайте случайным образом установим половину выходных значений на 0, а затем многократно повторим процесс с различными случайными масками, чтобы получить коллекцию различных предсказаний. Разброс предсказанных значений дает довольно хорошую оценку эпистемической неопределенности.

Стоит отметить, что выбор архитектуры модели влияет на компромисс между двумя рассмотренными типами неопределенности. Если вы используете масштабную модель с большим количеством параметров, то можете добиться, чтобы она очень точно соответствовала обучающим данным. Однако эта модель, вероятно, будет недообучена, поэтому множество различных комбинаций входных данных будет одинаково хорошо (но не идеально) соответствовать обучающим

данным. Если вместо этого вы используете небольшую модель с несколькими параметрами, то, скорее всего, найдется какой-то уникальный набор оптимальных значений параметров, при котором модель выдаст прогноз со стопроцентной вероятностью. Но, вероятно, такой набор не будет соответствовать обучающему набору и вашим ожиданиям. В любом случае, оба типа неопределенности должны быть включены в оценку точности прогнозов модели.

Звучит довольно сложно, не так ли? К счастью, DeepChem очень легко реализует оценку неопределенности. Просто добавьте один дополнительный аргумент в конструктор модели:

```
model = GraphConvModel(n_tasks=1, mode='regression', dropout=0.2, uncertainty=True)
```

Включая опцию `uncertainty=True`, мы добавляем в модель дополнительные выходы для неопределенности и вносим необходимые изменения в функцию потерь. Теперь мы можем делать прогнозы наподобие этого:

```
y_pred, y_std = model.predict_uncertainty(test_dataset)
```

Он многократно вычисляет вывод модели с разными масками исключения, а затем возвращает среднее значение и оценку стандартного отклонения для каждого элемента вывода.

На рис. 10.8 показано, как это работает на тестовом наборе. Для каждой выборки мы изображаем на графике фактическую ошибку прогноза в сравнении с прогнозируемой оценкой неопределенности модели. Данные показывают четкую тенденцию: выборки с большой прогнозируемой неопределенностью, как правило, сопровождаются более крупными ошибками, чем выборки с небольшой прогнозируемой неопределенностью. Пунктирная линия соответствует уравнению $y = 2x$. Точки ниже этой линии имеют прогнозируемые растворимости, которые находятся в пределах двух (прогнозируемых) стандартных отклонений от истинного значения. В этом регионе находятся примерно 90 % выборок.

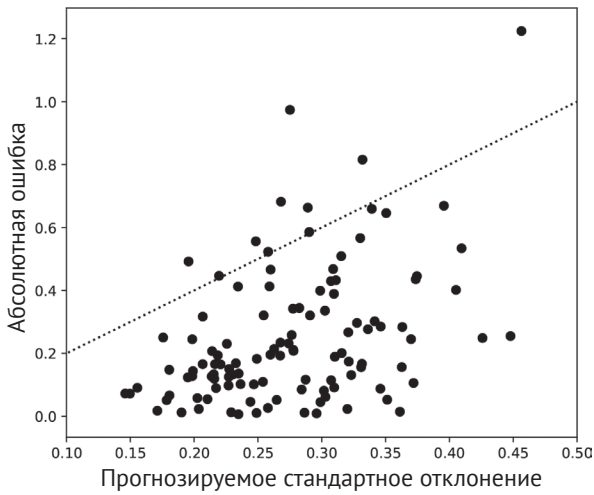


Рис. 10.8 ❖ Истинная ошибка в предсказаниях модели в сравнении с ее оценками неопределенности в каждом значении

Интерпретируемость, объяснимость и последствия для реального мира

Чем серьезнее последствия неправильного прогноза, тем важнее понять, как работает модель. Для некоторых моделей отдельные прогнозы не важны. Химик, работающий на ранних стадиях разработки лекарств, может использовать модель для скрининга миллионов потенциальных соединений и выбора наиболее перспективных претендентов для синтеза. Точность предсказаний модели может быть низкой, но это приемлемо. Пока подходящие соединения в среднем лучше отбракованных, они служат полезной цели.

В других случаях имеет значение каждый прогноз. Когда модель используется для диагностики заболевания или рекомендации лечения, от точности каждого результата может в буквальном смысле зависеть жизнь пациента. Вопрос «Должен ли я доверять этому результату?» становится жизненно важным.

В идеале модель должна давать не только диагноз, но и краткое изложение фактов, подтверждающих этот диагноз. Врач может изучить доказательства и принять обоснованное решение о том, правильно ли сработала модель в данном конкретном случае. Модель, обладающая этим свойством, называется *объяснимой*.

К сожалению, слишком много моделей глубокого обучения невозможно объяснить. В этом случае врач сталкивается с трудным выбором. Должны ли они доверять модели, даже если не знают, на каких доказательствах основан результат? Или они должны игнорировать модель и полагаться на собственное суждение? Ни один из вариантов не является удовлетворительным.

Запомните простой принцип: *каждая модель в конечном итоге взаимодействует с людьми*. Чтобы оценить качество модели, вы должны включить эти взаимодействия в свой анализ. Часто они зависят как от психологии или экономики, так и от машинного обучения. Недостаточно рассчитать коэффициент корреляции или ROC AUC для прогнозов модели. Вы должны подумать, *кто* увидит эти прогнозы, *как* они будут интерпретированы и *какие* реальные последствия будут иметь в конечном итоге.

Создание более интерпретируемой или объяснимой модели может не повлиять на точность ее прогнозов, но способно оказать огромное влияние на реальные *последствия* этих прогнозов. Это неотъемлемая часть процесса разработки модели.

Заключение

Глубокие модели имеют репутацию таинственных «черных ящиков», но в последнее время было разработано много полезных методов, которые помогают понять, чему на самом деле научилась ваша модель и как она работает. Исходя из этого понимания, вы можете решить, стоит ли доверять модели, и определить ситуации, в которых модель может потерпеть неудачу. Кроме того, перед вами может раскрыться новое понимание данных. Например, анализируя модель связывания, мы обнаружили мотив связывания для конкретного фактора транскрипции.

Глава 11

Практический пример виртуального скрининга

Виртуальный скрининг (virtual screening) может обеспечить эффективный и экономически выгодный способ определения отправных точек для поиска лекарственных соединений. Вместо того чтобы проводить дорогой экспериментальный *высокопроизводительный скрининг* (high-throughput screen, HTS), мы можем применить вычислительные методы для виртуальной оценки миллионов или даже десятков миллионов молекул. Методы виртуального скрининга часто разделяют на две категории: виртуальный скрининг на основе структуры и виртуальный скрининг на основе лигандов.

В виртуальном скрининге на основе структуры вычислительные методы применяются для поиска молекул, оптимально входящих в полость в структуре белка, известную как сайт связывания. Связывание молекулы с белком зачастую *ингибирует* (подавляет) основную функцию белка. Например, белки, известные как ферменты, провоцируют и ускоряют различные химические процессы в организме. Обнаружив и оптимизировав ингибиторы этих ферментативных процессов, ученые смогли разработать методы лечения широкого спектра заболеваний в онкологии, иммунологии и других терапевтических областях.

В виртуальном скрининге на основе лигандов мы ищем молекулы, которые функционируют аналогично одной или нескольким известным молекулам. Например, мы стремимся улучшить функцию существующей молекулы, избежать фармакологических проблем, связанных с известной молекулой, или разработать новую интеллектуальную собственность. Виртуальный скрининг на основе лигандов обычно начинается с формирования обучающего набора известных молекул, обнаруженных любым из множества экспериментальных методов. Затем на этом наборе обучается модель для виртуального скрининга и поиска новых отправных точек в синтезе лекарственных соединений.

В этой главе приведен практический пример рабочего процесса виртуального скрининга. Мы рассмотрим код, реализующий компоненты виртуального скрининга, а также рассуждения, лежащие в основе принятых решений. В данном случае мы будем выполнять виртуальный скрининг на основе лигандов. Для обучения сверточной нейронной сети мы воспользуемся двумя наборами молекул. О молекулах из первого набора известно, что они связываются с конкретным белком, а молекулы из второго набора предположительно не связываются с этим белком. Наша задача – обнаружить новые молекулы, обладающие потенциалом связывания с белковой мишенью.

Подготовка набора данных для прогнозного моделирования

В качестве первого шага мы создадим графовую сверточную модель для предсказания способности молекул ингибировать белок, известный как ERK2. Этот белок, также известный как активируемая митогеном протеинкиназа 1 (mitogen-activated protein kinase 1, MAPK1), играет важную роль в сигнальных механизмах, регулирующих размножение клеток. ERK2 участвует в ряде раковых заболеваний, а ингибиторы ERK2 в настоящее время проходят клинические испытания на способность сдерживать немелкоклеточный рак легких и меланому (рак кожи).

Мы будем обучать модель отличать набор соединений, активных в отношении ERK2, от набора бесполезных (неактивных) соединений. Наборы активных и бесполезных соединений извлечены из базы данных DUD-E, предназначенной для тестирования прогностических моделей. На практике мы обычно получаем наборы активных и бесполезных молекул из научной литературы или из базы данных биологически активных молекул, такой как база данных ChEMBL от Европейского института биоинформатики (European Bioinformatics Institute). Для создания точной модели нам следует иметь набор бесполезных соединений с распределением свойств, аналогичным распределению свойств активных соединений. Допустим, что это не так, и в обучающем наборе молекулярная масса неактивных соединений всегда меньше, чем у активных соединений. В этом случае наш классификатор научится просто разделять низкомолекулярные и высокомолекулярные соединения. От такого классификатора на практике будет мало пользы.

Чтобы лучше изучить набор данных, давайте рассмотрим несколько вычисляемых свойств наших активных и бесполезных молекул. Чтобы построить надежную модель, мы должны убедиться, что свойства активных молекул аналогичны свойствам молекул-пустышек.

Во-первых, давайте импортируем необходимые библиотеки:

```
from rdkit import Chem             # Библиотеки RDKit для химических функций
from rdkit.Chem import Draw        # Рисование химических структур
import pandas as pd                # Работа с табличными данными
from rdkit.Chem import PandasTools # Работа с химическими данными
from rdkit.Chem import Descriptors # Вычисление дескрипторов молекул
from rdkit.Chem import rdmolops    # Дополнительные свойства молекул
import seaborn as sns              # Построение графиков
```

В этом примере молекулы представлены с помощью строк SMILES. Для получения дополнительной информации о SMILES см. главу 4. Теперь мы можем прочитать файл SMILES в кадр данных Pandas и добавить в этот кадр данных молекулу RDKit. Хотя входной файл SMILES технически не является CSV-файлом, функция Pandas `read_CSV()` может читать его, если указан разделитель, который в данном случае является пробелом:

```
active_df = pd.read_CSV("mk01/actives_final.ism",header=None,sep=" ")
active_rows,active_cols = active_df.shape
active_df.columns = ["SMILES","ID","ChEMBL_ID"]
active_df["label"] = ["Active"]*active_rows
PandasTools.AddMoleculeColumnToFrame(active_df,"SMILES","Mol")
```

Определим функцию для добавления вычисляемых свойств в кадр данных:

```
def add_property_columns_to_df(df_in):
df_in["mw"] = [Descriptors.MolWt(mol) for mol in df_in.Mol]
df_in["logP"] = [Descriptors.MolLogP(mol) for mol in df_in.Mol]
df_in["charge"] = [rdmolops.GetFormalCharge(mol) for mol in df_in.Mol]
```

С помощью этой функции мы можем рассчитать молекулярную массу, LogP и *формальный заряд* (formal charge) молекул. Эти свойства описывают размер молекулы, ее коэффициент распределения в системе октанол–вода и положительный или отрицательный заряд молекулы. Как только мы получим эти свойства, мы сможем сравнить распределения для активных и неактивных наборов:

```
add_property_columns_to_df(active_df)
```

Давайте взглянем на первые несколько строк нашего кадра данных, чтобы убедиться, что содержимое кадра соответствует входному файлу (табл. 11.1):

```
active_df.head()
```

Таблица 11.1. Первые несколько строк кадра данных `active_df`

	Строка SMILES	ID	ChEMBL_ID	Метка
0	Cn1ccnc1Sc2ccc(cc2Cl)Nc3c4cc(c(cc4ncc3C#N)OCCCN5CCOCC5)OC	168691	CHEMBL318804	Active
1	C[C@@]12[C@@H]([C@@H](CC(O1)n3c4ccccc4c5c3c6n2c7ccccc7c6c8c5C(=O)NC8)NC)OC	86358	CHEMBL162	Active
2	Cc1cnc(nc1c2cc([nH]c2)C(=O)N[C@H](CO)c3cccc(c3)Cl)Nc4cccc5c4OC(O5)(F)F	575087	CHEMBL576683	Active
3	Cc1cnc(nc1c2cc([nH]c2)C(=O)N[C@H](CO)c3cccc(c3)Cl)Nc4cccc5c4OCO5	575065	CHEMBL571484	Active
4	Cc1cnc(nc1c2cc([nH]c2)C(=O)N[C@H](CO)c3cccc(c3)Cl)Nc4cccc5c4CCC5	575047	CHEMBL568937	Active

Теперь сделаем то же самое с молекулами-пустышками:

```
decoy_df = pd.read_CSV("mk01/decoys_final.ism",header=None,sep=" ")
decoy_df.columns = ["SMILES","ID"]
decoy_rows, decoy_cols = decoy_df.shape
decoy_df["label"] = ["Decoy"]*decoy_rows
PandasTools.AddMoleculeColumnToFrame(decoy_df,"SMILES","Mol")
add_property_columns_to_df(decoy_df)
```

Чтобы построить модель, нам нужен единый кадр данных с активными и бесполезными молекулами. Мы можем использовать функцию добавления Pandas, чтобы добавить два кадра данных и создать новый кадр с именем tmp_df:

```
tmp_df = active_df.append(decoy_df)
```

Для сравнения вычисляемых свойств активного и бесполезного наборов молекул мы будем использовать так называемые *скрипичные графики* (violin plot). Скрипичные графики аналогичны *коробочным диаграммам* (boxplot). Скрипичный график обеспечивает зеркальное горизонтальное представление распределения частотности. В идеале мы хотели бы видеть одинаковые распределения для активного и бесполезного наборов. Результаты показаны на рис. 11.1:

```
sns.violinplot(tmp_df["label"],tmp_df["mw"])
```

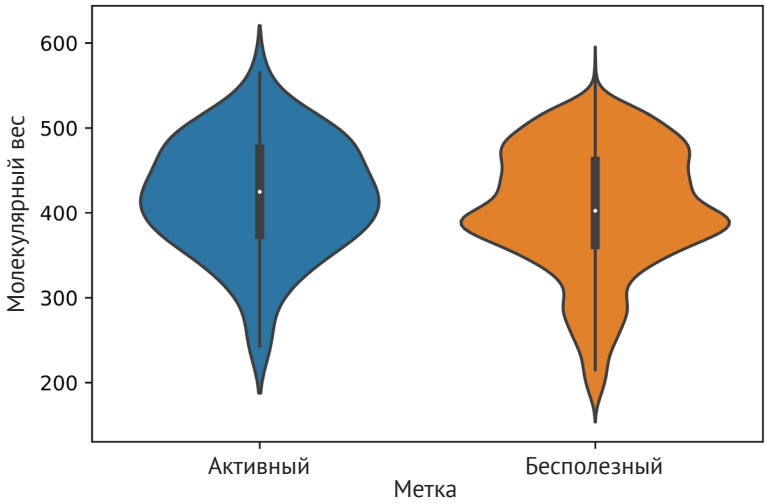


Рис. 11.1 ❖ Скрипичные графики распределения молекулярной массы для активных и бесполезных наборов

Изучение этих графиков показывает, что распределения молекулярных масс для двух наборов приблизительно эквивалентны. Набор «пустышек» содержит более низкомолекулярные соединения, но центр распределения, показанный в виде прямоугольника в середине каждого скрипичного участка, находится в одинаковом месте на обоих графиках.

Аналогичным образом сравним скрипичные графики распределения LogP (рис. 11.2). Опять же, мы можем видеть, что распределения похожи, с несколькими дополнительными молекулами-пустышками на нижнем конце распределения:

```
sns.violinplot(tmp_df["label"],tmp_df["logP"])
```

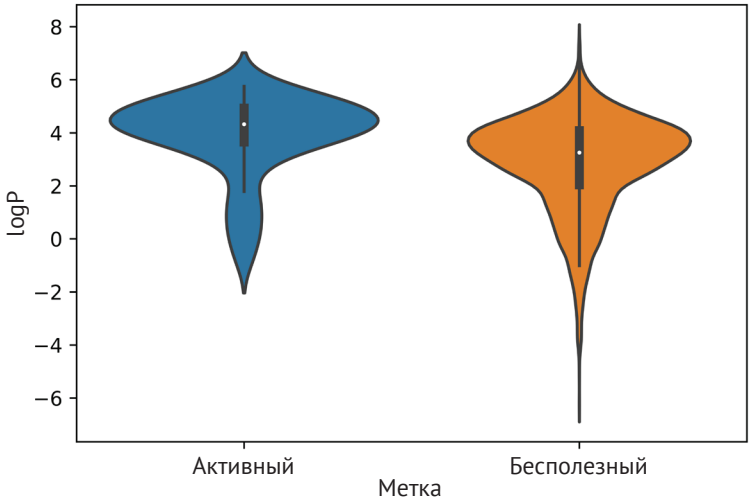


Рис. 11.2 ❖ Скрипичные графики распределения LogP для активных и бесполезных наборов

Наконец, мы выполняем такое же сравнение формальных зарядов молекул (рис. 11.3):

```
sns.violinplot(new_tmp_df["label"],new_tmp_df["charge"])
```

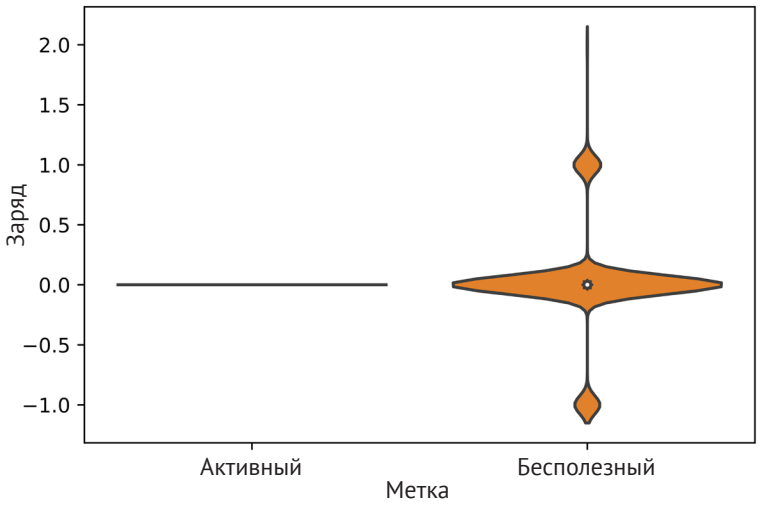


Рис. 11.3 ❖ Скрипичные графики распределения формального заряда для активных и бесполезных наборов

В этом случае мы видим существенную разницу. Все активные молекулы являются нейтральными (имеют заряд 0), в то время как некоторые из бесполезных молекул имеют заряд +1 или –1. Давайте посмотрим, какая часть молекул-пустышек заряжена. Мы можем сделать это, создав новый кадр данных только с заряженными молекулами:

```
charged = decoy_df[decoy_df["charge"] != 0]
```

У кадра данных Pandas есть свойство `shape`, которое возвращает количество строк и столбцов в кадре данных. Таким образом, `element[0]` в свойстве `shape` будет количеством строк. Давайте разделим количество строк в нашем кадре заряженных молекул на общее количество строк в кадре бесполезных данных:

```
charged.shape[0]/decoy_df.shape[0]
```

Эта операция возвращает значение 0,162. Как мы и видели на скрипичном графике, примерно 16 % молекул-пустышек имеют заряд. Похоже, это связано с тем, что активные и бесполезные наборы не были подготовлены по согласованной схеме. Мы можем решить эту проблему, изменив химическую структуру молекул-пустышек, чтобы нейтрализовать их заряды. К счастью, мы можем легко сделать это с помощью функции `NeutraliseCharges()`, описанной в RDKit Cookbook:

```
from neutralize import NeutraliseCharges
```

Чтобы избежать путаницы, мы создадим новый кадр данных со строками, идентификаторами и метками SMILES для молекул-пустышек:

```
revised_decoy_df = decoy_df[["SMILES","ID","label"]].copy()
```

Имея новый кадр данных, мы можем заменить исходные строки SMILES на строки с нейтральными формами молекул. Функция `NeutraliseCharges` возвращает два значения. Первое – строка SMILES для нейтральной формы молекулы, а второе – булева переменная, указывающая, была ли молекула изменена. В следующем коде нам нужна только строка SMILES, поэтому мы используем первый элемент кортежа, возвращаемый `NeutraliseCharges`:

```
revised_decoy_df["SMILES"] = [NeutraliseCharges(x)[0] for x
in revised_decoy_df["SMILES"]]
```

После того как мы заменили строки SMILES, мы можем добавить столбец молекулы в наш новый кадр данных и снова вычислить свойства:

```
PandasTools.AddMoleculeColumnToFrame(revised_decoy_df,"SMILES","Mol")
add_property_columns_to_df(revised_decoy_df)
```

Затем мы можем добавить кадр данных с активными молекулами к исправленному набору бесполезных молекул:

```
new_tmp_df = active_df.append(revised_decoy_df)
```

Теперь мы можем построить новый график, чтобы сравнить распределение заряда активных молекул с распределением наших нейтрализованных пустышек (рис. 11.4):

```
sns.violinplot(new_tmp_df["label"],new_tmp_df["charge"])
```

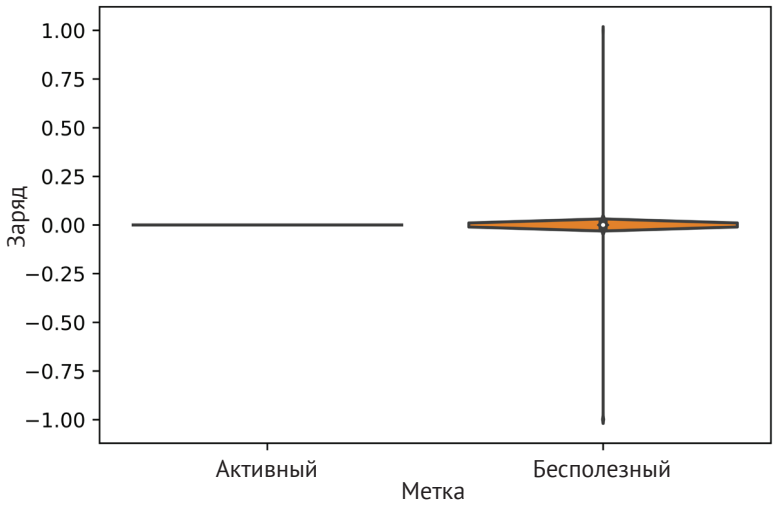


Рис. 11.4 ❖ Скрипичные графики распределения заряда для пересмотренного набора бесполезных молекул

Изучение графиков показывает, что в наборе молекул-пустышек сейчас очень мало заряженных молекул. Мы можем использовать тот же подход, что и ранее, чтобы создать кадр данных только с заряженными молекулами. Затем мы используем этот кадр, чтобы определить количество заряженных молекул, оставшихся в наборе:

```
charged = revised_decoy_df[revised_decoy_df["charge"] != 0]
charged.shape[0]/revised_decoy_df.shape[0]
```

Сейчас результат составляет 0,003. Мы сократили долю заряженных молекул с 16 % до 0,3 %. Теперь мы можем быть уверены, что наши наборы достаточно хорошо сбалансированы.

Чтобы использовать эти наборы данных с пакетом DeepChem, нам нужно записать молекулы в виде файла CSV, содержащего для каждой молекулы строку SMILES, ID, имя и целочисленное значение, указывающее, является ли соединение активным (помечено как 1) или неактивным (помечено как 0):

```
active_df["is_active"] = [1] * active_df.shape[0]
revised_decoy_df["is_active"] = [0] * revised_decoy_df.shape[0]
combined_df = active_df.append(revised_decoy_df)[["SMILES","ID","is_active"]]
combined_df.head()
```

Первые пять строк показаны в табл. 11.2.

Таблица 11.2. Первые несколько строк нового комбинированного кадра данных

	SMILES	ID	Активность (is_active)
0	Cn1 ccnc1Sc2ccc(cc2Cl}Nc3c4cc(c(cc4ncc3C#N}OCCCN5CCOCC5)OC	168691	1
1	C[C@@]12[C@@H]([C@@H](CC(O1)n3c4ccccc4c5c3c6n2c7ccccc7c6c8c5C(=O)NC8)NC)OC	86358	1
2	Cc1cnc(nc1c2cc([nH]c2)C(=0) N[C@H](CO)c3cccc(c3}Cl}Nc4cccc5c4OC(O5)(F)F	575087	1
3	CCc1cnc(nc1c2cc([nH]c2)C(=O)N[C@H](CO)c3cccc(c3}Cl}Nc4cccc5c4OCO5	575065	1
4	Cc1cnc(nc1c2cc([nH]c2)C(=0) N[C@H](CO)c3cccc(c3}Cl}Nc4cccc5c4CCC5	575047	1

Наш последний шаг на этом этапе – сохранить наш новый `combined_df` в виде файла CSV. Опция `index=False` заставляет Pandas не включать номер строки в первый столбец:

```
combined_df.to_csv ( "dude_erk1_mk01.CSV", индекс = False)
```

Обучение прогностической модели

Теперь, когда мы позаботились о форматировании, мы можем использовать эти данные для обучения графовой сверточной модели. Сначала нам нужно импортировать необходимые библиотеки. Некоторые из этих библиотек уже были импортированы в предыдущих примерах кода, но давайте предположим, что мы начинаем работу с CSV-файла, созданного в предыдущем разделе:

```
import deepchem as dc                        # Библиотеки DeepChem
from deepchem.models import GraphConvModel # Графовая свертка
import numpy as np                           # Библиотека NumPy для вычислений
import sys                                   # Обработка ошибок
import pandas as pd                          # Работа с таблицами данных
import seaborn as sns                        # Библиотека Seaborn для графиков
from rdkit.Chem import PandasTools           # Химические структуры в Pandas
```

Теперь определим функцию для создания `GraphConvModel`. В данном случае мы будем создавать классифицирующую модель. Поскольку позже мы применим мо-

дель к другому набору данных, хорошая идея – создать каталог для хранения модели. Вам нужно будет подставить имя каталога и путь в вашей файловой системе вместо фрагмента, выделенного жирным шрифтом:

```
def generate_graph_conv_model():
batch_size = 128
model = GraphConvModel(1, batch_size=batch_size,
mode='classification',
model_dir="/tmp/mk01/model_dir")
return model
```

Чтобы обучить модель, мы сначала читаем содержимое файла CSV, созданного в предыдущем разделе:

```
dataset_file = "dude_erk2_mk01.CSV"
tasks = ["is_active"]
featurizer = dc.feat.ConvMolFeaturizer()
loader = dc.data.CSVLoader(tasks=tasks,
smiles_field="SMILES",
featurizer=featurizer)
dataset = loader.featurize(dataset_file, shard_size=8192)
```

Теперь, когда у нас есть загруженный набор данных, давайте построим модель. Мы создадим обучающий и проверочный наборы для оценки точности модели. В этом случае мы будем использовать разделитель наборов `RandomSplitter` (DeepChem также предлагает ряд других разделителей наборов данных, таких как `ScaffoldSplitter`, который делит набор данных по химическому строению, и `ButinaSplitter`, который сначала кластеризует данные, а затем разделяет набор данных таким образом, что в учебном и проверочном наборах оказываются разные кластеры):

```
splitter = dc.splits.RandomSplitter()
```

С помощью разделения набора данных мы можем обучить модель на обучающем наборе, а затем протестировать ее на проверочном наборе. Теперь нам нужно выбрать оценочные метрики и оценить точность нашей модели. Имейте в виду, что наш набор данных не сбалансирован – у нас есть небольшое количество активных соединений и большое количество неактивных соединений. Учитывая этот дисбаланс, нам нужно использовать метрику, которая отражает точность на несбалансированных наборах данных. Одной из метрик, которая подходит для таких наборов данных, является *коэффициент корреляции Мэтьюса* (Matthews correlation coefficient, MCC):

```
metrics = [
dc.metrics.Metric(dc.metrics.matthews_corrcoef, np.mean,
mode="classification")]
```

Чтобы оценить точность нашей модели, мы выполним 10-кратную перекрестную проверку, обучая модель на обучающем наборе и, соответственно, проверяя на проверочном наборе:

```
training_score_list = []
validation_score_list = []
transformers = []
```

```
cv_folds = 10
for i in range(0, cv_folds):
model = generate_graph_conv_model()
res = splitter.train_valid_test_split(dataset)
train_dataset, valid_dataset, test_dataset = res
model.fit(train_dataset)
train_scores = model.evaluate(train_dataset, metrics,
transformers)
training_score_list.append(
train_scores["mean-matthews_corrcoef"])
validation_scores = model.evaluate(valid_dataset,
metrics,
transformers)
validation_score_list.append(
validation_scores["mean-matthews_corrcoef"])
print(training_score_list)
print(validation_score_list)
```

Чтобы визуализировать точность нашей модели на обучающих и тестовых данных, мы можем построить коробчатые диаграммы. Результаты показаны на рис. 11.5:

```
sns.boxplot(
["training"] * cv_folds + ["validation"] * cv_folds,
training_score_list + validation_score_list)
```

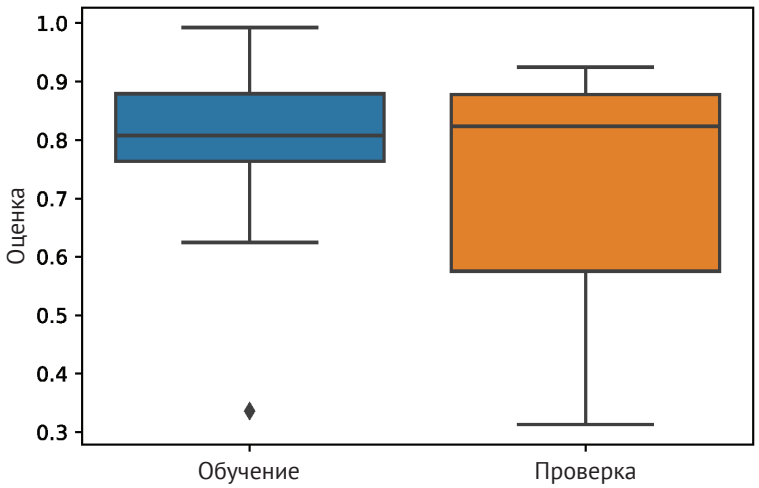


Рис. 11.5 ❖ Коробчатые диаграммы для обучающих и проверочных наборов

Графики показывают, что, как и ожидалось, точность на обучающем наборе превосходит точность на проверочном наборе. Тем не менее точность на проверочном наборе все еще довольно хорошая. На данный момент мы можем быть уверены в точности нашей модели.

Было бы полезно визуализировать результаты нашей модели. Для этого мы сгенерируем набор прогнозов для проверочного набора:

```
pred = [x.flatten() for x in model.predict(valid_dataset)]
```

Чтобы упростить обработку, мы создадим кадр данных Pandas с прогнозами:

```
pred_df = pd.DataFrame(pred,columns=["neg","pos"])
```

Мы можем с легкостью добавить в кадр данных класс активности (1 = активный, 0 = неактивный) и строки SMILES для наших предсказанных молекул:

```
pred_df["active"] = [int(x) for x in valid_dataset.y]
pred_df["SMILES"] = valid_dataset.ids
```

Всегда полезно взглянуть на первые несколько строк кадра данных, чтобы убедиться, что данные имеют смысл. В табл. 11.3 представлен пример таких строк.

Таблица 11.3. Первые несколько строк кадра данных с предсказаниями

	neg	pos	active	
0	0.906081	0.093919	1	Cn1ccnc1Sc2ccc(cc2Cl)Nc3c4cc(c(cc4ncc3C#N)OCCC...
1	0.042446	0.957554	1	Cc1cnc(nc1c2cc([nH]c2)C(=O)N[C@H](CO)c3cccc(c3...
2	0.134508	0.865492	1	Cc1cccc(c1)[C@@H](CO)NC(=O)c2cc(c[nH]2)c3c(cnc...
3	0.036508	0.963492	1	Cc1cnc(nc1c2cc([nH]c2)C(=O)N[C@H](CO)c3ccccc3)...
4	0.940717	0.059283	1	c1c\2c([nH]c1Br)C(=O)NCC/C2=C/3\C(=O)N=C(N3)N

Коробчатые диаграммы помогут нам сравнить предсказанные значения для активных и неактивных молекул (рис. 11.6).

```
sns.boxplot(pred_df.active,pred_df.pos)
```

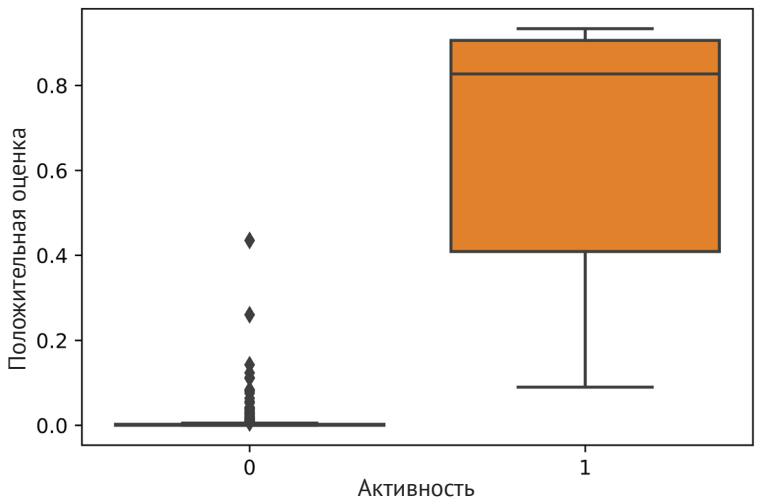


Рис. 11.6 ❖ Положительные оценки для предсказанных молекул

У нашей модели очень хорошая точность: мы видим четкое разделение между активными и неактивными молекулами. При построении прогнозирующей модели часто важно исследовать неактивные молекулы, которые прогнозируются как активные (ложноположительные результаты), а также активные молекулы, которые прогнозируются как неактивные (ложноотрицательные результаты). Похоже, что только одна из наших активных молекул получила низкий положительный

балл. Чтобы рассмотреть этот момент получше, мы создадим новый кадр данных, содержащий все активные молекулы с положительной оценкой < 0,5:

```
false_negative_df = pred_df.query("active == 1 & pos < 0.5").copy()
```

Для изучения химической структуры молекул в нашем кадре данных мы используем модуль PandasTools из RDKit:

```
PandasTools.AddMoleculeColumnToFrame(false_negative_df,
"SMILES", "Mol")
```

Давайте посмотрим на новый кадр данных (рис. 11.7):

```
false_negative_df
```

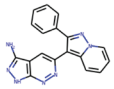
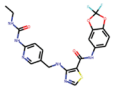
	Отрицательный	Положительный	Активность	SMILES	Молекула
4	0.723421	0.27658	1	c1ccc(cc1)c2c(c3ccccn3n2)c4cc5c(n[nH]c5nn4)N	
5	0.910040	0.08996	1	CCNC(=O)Nc1ccc(cn1)CNc2c(scn2)C(=O)Nc3ccc4c(c3)OC(O4)(F)F	

Рис. 11.7 ❖ Ложноотрицательные прогнозы

Чтобы в полной мере использовать информацию, содержащуюся в этом кадре данных, нам необходимо иметь некоторые знания в области медицинской химии. Иногда помогает сравнение химической структуры ложноотрицательных молекул с химической структурой истинно положительных молекул. Это может дать некоторое представление о причинах ошибочного прогноза. Часто бывает так, что ложноотрицательные молекулы просто не похожи ни на одну из истинно положительных молекул. В этом случае, возможно, стоит расширить область литературного поиска, чтобы увеличить разнообразие молекул в обучающем наборе.

Мы можем использовать аналогичный подход для изучения ложноположительных молекул, которые неактивны, но получили положительный результат `pos > 0,5` (рис. 11.8). Опять же, иногда помогает сравнение с химической структурой истинно положительных молекул:

```
false_positive_df = pred_df.query("active == 0 & pos > 0.5").copy()
PandasTools.AddMoleculeColumnToFrame(false_positive_df, "SMILES", "Mol")

false_positive_df
```

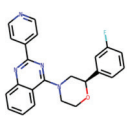
	Отрицательный	Положительный	Активность	SMILES	Молекула
296	0.564975	0.435025	0	c1ccc2c(c1)c(nc(n2)c3ccncc3)N4CCO[C@@H]c5cccc(c5)F	

Рис. 11.8 ❖ Ложноположительная молекула

На этапе обучения наша цель заключалась в оценке точности модели. Таким образом, мы обучили модель на части данных и проверили модель на оставшейся части. Теперь пришло время создать наиболее точную модель. Для этого мы обучим модель на всех данных:

```
model.fit(dataset)
```

Это дает нам показатель точности 91 %. Наконец, мы сохраняем модель на диск, чтобы могли использовать ее для будущих прогнозов:

```
model.save()
```

Подготовка набора данных для прогнозирования

После создания и проверки прогностической модели мы можем применить ее к новому набору молекул. Во многих случаях прогностическую модель создают на основе литературных данных, а затем применяют эту модель к набору молекул, нуждающихся в проверке на активность. Проверяемые молекулы могут поступать из внутренней базы данных или из коммерческой коллекции скрининга. В качестве примера мы будем использовать созданную нами прогностическую модель для скрининга небольшой выборки из 100 000 соединений, извлеченной из базы данных ZINC. Эта база данных представляет собой коммерческую коллекцию более чем из одного миллиарда молекул.

Одной из потенциальных проблем при проведении виртуального скрининга является присутствие молекул, которые могут помешать измерению биологической активности. За последние 25 лет многие исследовательские группы разработали набор правил для идентификации потенциально реактивных или проблемных молекул. Несколько таких наборов правил, закодированных в виде строк SMARTS, были собраны группой, которая курирует базу данных ChEMBL. Эти наборы правил распространяются в виде скрипта Python, который называется rd_filters.py. Мы выспользуемся скриптом rd_filters.py для выявления потенциально проблемных молекул в нашем наборе из 100 000 молекул из базы данных ZINC.

Файл rd_filters.py и связанные файлы данных доступны в репозитории GitHub для этой книги.

Чтобы ознакомится с доступными режимами и аргументами скрипта, вызовите его с флагом -h:

```
rd_filters.py -h

Usage:
rd_filters.py $ filter --in INPUT_FILE --prefix PREFIX [--rules RULES_FILE_NAME]
[--alerts ALERT_FILE_NAME][--np NUM_CORES]
rd_filters.py $ template --out TEMPLATE_FILE [--rules RULES_FILE_NAME]

Options:
--in INPUT_FILE input file name
--prefix PREFIX prefix for output file names
--rules RULES_FILE_NAME name of the rules JSON file
--alerts ALERTS_FILE_NAME name of the structural alerts file
--np NUM_CORES the number of cpu cores to use (default is all)
--out TEMPLATE_FILE parameter template file name
```

Вызывая скрипт для входного файла zinc_100k.smi, мы можем указать не только имя входного файла, но и префикс для имен выходных файлов. Аргумент `filter` вызывает скрипт в режиме «фильтра», когда он обнаруживает потенциально проблемные молекулы. Аргумент `--prefix` указывает, что имена выходных файлов будут начинаться с префикса `zinc`.

```
rd_filters.py filter --in zinc_100k.smi --prefix zinc

using 24 cores
Using alerts from Inpharmatica
Wrote SMILES for molecules passing filters to zinc.smi
Wrote detailed data to zinc.CSV
68752 of 100000 passed filters 68.8%
Elapsed time 15.89 seconds
```

Из выходных данных мы можем извлечь следующую информацию:

- скрипт работает на 24 ядрах. Число ядер можно выбрать с помощью флага `-np`;
- скрипт использует набор правил Inpharmatica. Этот набор правил охватывает широкий спектр химических свойств, которые, как было обнаружено, доставляют проблемы при биологических исследованиях. В дополнение к набору Inpharmaticia в скрипте доступно семь других наборов правил. Пожалуйста, ознакомьтесь с документацией к скрипту rd_filters.py для получения дополнительной информации;
- строки SMILES для молекул, прошедших через фильтры, были записаны в файл с именем zinc.smi. Мы будем использовать этот файл в качестве входных данных для прогностической модели;
- подробная информация о том, какие соединения вызвали определенные предупреждения, была записана в файл с именем zinc.CSV;
- 69 % молекул прошли фильтры, а 31 % были признаны проблемными.

Взглянув на причины, по которым молекулы были отклонены, мы можем понять, нужно ли нам скорректировать настройки какого-либо фильтра. Для просмотра первых нескольких строк вывода, показанных в табл. 11.4, достаточно короткого кода Python:

```
import pandas as pd
df = pd.read_CSV("zinc.CSV")
df.head()
```

Таблица 11.4. Первые несколько строк кадра данных, созданного из файла zinc.CSV

SMILES	NAME	FILTER	MW	LogP	HBD
CN(CCO)C[C@@H](O)Cn1cnc2c1c(=O)n(C)c(=O)n2C	ZINC000000000843	Filter82_pyridinium >0	311.342	−2.2813	2
O=c1[nH]c(=O)n([C@@H]2C[C@@H](O)[C@H](CO)O2)cc1Br	ZINC000000001063	Filter82_pyridinium >0	307.100	−1.0602	3
Cn1c2ncn(CC(=O)N3CCOCC3)c2c(=O)n(C)c1=O	ZINC000000003942	Filter82_pyridinium >0	307.310	−1.7075	0
CN1C(=O)C[C@H](N2CCN(C(=O)CN3CCCC3)CC2)C1=O	ZINC000000036436	OK	308.382	−1.0163	0
CC(=O)NC[C@H](O)[C@H]1O[C@H]2OC(C)(C)O[C@H]2[C...	ZINC000000041101	OK	302.327	−1.1355	3

Кадр данных имеет шесть столбцов:

- **SMILES** – строки SMILES для каждой молекулы;
- **NAME** – имя молекулы, как указано во входном файле;
- **FILTER** – причина, по которой молекула была отклонена, или **OK**, если молекула не была отклонена;
- **MW** – молекулярный вес молекулы. По умолчанию молекулы с молекулярной массой более 500 отклоняются;
- **LogP** – рассчитанный коэффициент распределения в системе октанол–вода. По умолчанию молекулы с LogP > 5 отклоняются;
- **HBD** – число доноров водородных связей. По умолчанию молекулы с HBD > 5 отклоняются.

Мы можем использовать класс `Counter` из библиотеки `collections`, чтобы определить, какие фильтры были ответственны за удаление наибольшего числа молекул (табл. 11.5):

```
from collections import Counter
count_list = list(Counter(df.FILTER).items())
count_df = pd.DataFrame(count_list,columns=["Rule","Count"])
count_df.sort_values("Count",inplace=True,ascending=False)
count_df.head()
```

Таблица 11.5. Количество молекул, отклоненных наиболее активными фильтрами

	Rule	Count
1	OK	69156
6	Filter41_12_dicarbonyl > 0	19330
0	Filter82_pyridinium > 0	7761
10	Filter93_acetyl_urea > 0	1541
11	Filter78_bicyclic_lmide > 0	825

Первая строка в таблице, помеченная как **OK**, указывает количество молекул, которые не были удалены ни одним из фильтров. Здесь видно, что 69 156 молекул входного набора прошли все фильтры. Наибольшее количество молекул (19 330) было отклонено, потому что они содержали 1,2-дикарбонильную группу. Молекулы этого типа способны реагировать и образовывать ковалентные связи с белковыми остатками, такими как серин и цистеин. Мы можем найти шаблон SMARTS, используемый для обнаружения этих молекул, поиском строки «Filter41_12_dicarbonyl» в файле filter_collection.CSV, который является частью дистрибутива rd_filters.py. Шаблон SMARTS имеет вид «** C (= O) C (= O) ** » и представляет в молекуле:

- любой атом, связанный с…;
- углерод с двойной связью с кислородом, связанный с…;
- углерод с двойной связью с кислородом, связанный с…;
- любой атом….

Всегда полезно внимательно посмотреть на данные и убедиться, что все работает как положено. Мы можем использовать аргумент `highlightAtomLists` для функции RDKit `MolsToGridImage()`, чтобы выделить признаки 1,2-дикарбонила (рис. 11.9):

```
from rdkit import Chem
from rdkit.Chem import Draw
```

```
mol_list = [Chem.MolFromSmiles(x) for x in smiles_list]
dicarbonyl = Chem.MolFromSmarts('*C(=O)C(=O)*')
match_list = [mol.GetSubstructMatch(dicarbonyl) for mol in mol_list]
Draw.MolsToGridImage(mol_list, highlightAtomLists=match_list, molsPerRow=5)
```

Теперь мы видим, что молекулы действительно имеют дикарбонильные группы, как показано на рисунке. Если бы мы захотели, то могли бы аналогичным образом проверить другие фильтры. На данный момент, однако, мы можем быть довольны результатами фильтрации. Мы удалили проблемные молекулы из набора, который планируем использовать для виртуального скрининга. Теперь можно перейти к следующему этапу нашего упражнения и воспользоваться набором, хранящимся в файле zinc.smi.

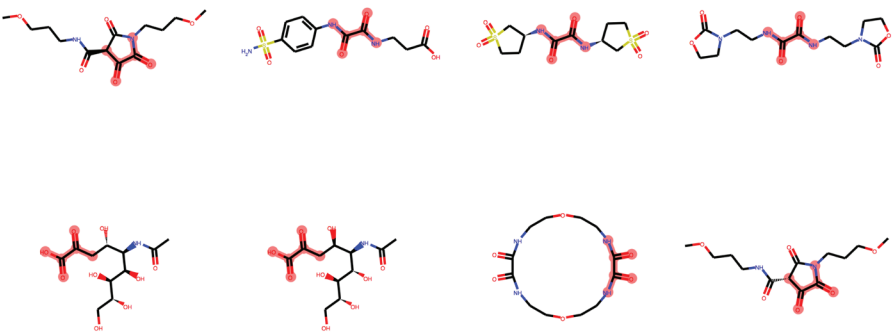


Рис. 11.9 ❖ Молекулы, содержащие 1,2-дикарбонильную группу

Применение прогностической модели

Воспользуемся ранее созданной моделью `GraphConvModel` для поиска набора коммерчески доступных соединений, которые мы только что отфильтровали. Применение модели осуществляется в несколько шагов.

1. Загрузите модель с диска.
2. Создайте фичеризатор.
3. Прочитайте и фичеризуйте молекулы, которые будут проходить через модель.
4. Изучите оценки прогнозов.
5. Изучите химические структуры молекул из верхушки рейтинга.
6. Кластеризуйте выбранные молекулы.
7. Запишите выбранные молекулы из каждого кластера в файл CSV.

Начнем с импорта необходимых библиотек:

```
import deepchem as dc                                # Библиотека DeepChem
import pandas as pd                                  # Библиотека Pandas для таблиц
from rdkit.Chem import PandasTools, Draw             # Chemistry в Pandas
from rdkit import DataStructs                        # Поддержка "отпечатков" молекул
from rdkit.ML.Cluster import Butina                  # Кластеры молекул
from rdkit.Chem import rdMolDescriptors as rdmd # Дескрипторы
import seaborn as sns                                # Рисование графиков
```

и загрузки модели, которую мы сгенерировали ранее:

```
model = dc.models.TensorGraph.load_from_dir(""/tmp/mk01/model_dir"")
```

Чтобы сгенерировать прогнозы из нашей модели, нам сначала нужно выполнить фичеризацию молекул, которые мы планируем использовать для генерации прогнозов. Мы делаем это путем создания экземпляра объекта DeepChem `ConvMolFeaturizer`:

```
featurizer = dc.feat.ConvMolFeaturizer()
```

Для фичеризации молекул следует преобразовать наш файл SMILES в файл CSV. Фичеризатор DeepChem требует наличия столбца активности, поэтому мы добавляем его и записываем файл в CSV:

```
df = pd.read_CSV("zinc.smi",sep=" ",header=None)
df.columns=["SMILES","Name"]
rows,cols = df.shape
# Добавляем пустой столбец, чтобы соответствовать требованиям фичеризатора
df["Val"] = [0] * rows
```

Как и прежде, мы должны проверить первые несколько строк файла (табл. 11.6), чтобы убедиться, что все в точности так, как мы ожидали:

```
df.head()
```

Таблица 11.6. Первые несколько строк входного файла

	SMILES	Name	Val
0	CN1C(=O)C[C@H](N2CCN(C(=O)CN3CCCC3)CC2)C1=O	ZINC000000036436	0
1	CC(=O)NC[C@H](O)[C@H]1O[C@H]2OC(C)(C)O[C@H]2[C@@H]1NC(C)=O	ZINC000000041101	0
2	C1CN(c2nc(-c3nn[nH]n3)nc(N3CCOCC3)n2)CCO1	ZINC000000054542	0
3	OCCN(CCO)c1nc(Cl)nc(N(CCO)CCO)n1	ZINC000000109481	0
4	COC(=O)c1ccc(S(=O)(=O)N(CCO)CCO)n1C	ZINC000000119782	0

Обратите внимание, что столбец Val – это просто заполнитель, удовлетворяющий требованиям фичеризатора DeepChem к формату файла. Содержимое файла выглядит хорошо, поэтому мы запишем его в виде файла CSV для использования в качестве входных данных DeepChem. Мы используем аргумент `index=False`, чтобы запретить Pandas писать числовой индекс в первом столбце:

```
infile_name = "zinc_filtered.CSV"
df.to_CSV(infile_name, index=False)
```

Воспользуемся возможностями пакета DeepChem, чтобы прочитать этот CSV-файл с помощью загрузчика и фичеризовать молекулы, с которыми планируем работать:

```
loader = dc.data.CSVLoader(tasks=['Val'], smiles_field="SMILES", featurizer=featurizer)
dataset = loader.featurize(infile_name, shard_size=8192)
```

Передадим фичеризованные молекулы в модель для генерации прогнозов:

```
pred = model.predict(dataset)
```

Для удобства поместим прогнозы в кадр данных Pandas:

```
pred_df = pd.DataFrame([x.flatten() for x in pred],
columns=["Neg", "Pos"]
```

График, доступный в библиотеке Seaborn, дает наглядную картину распределения результатов. К сожалению, в виртуальном скрининге нет четкого критерия для прекращения деятельности. Зачастую лучшая стратегия – посмотреть на распределение баллов, а затем выбрать молекулы с наибольшим количеством баллов. Если мы посмотрим на график на рис. 11.10, то увидим, что существует только небольшое количество молекул с показателями выше 0,3. Мы можем использовать это значение в качестве предварительного условия для отбора молекул, подлежащих экспериментальной проверке.

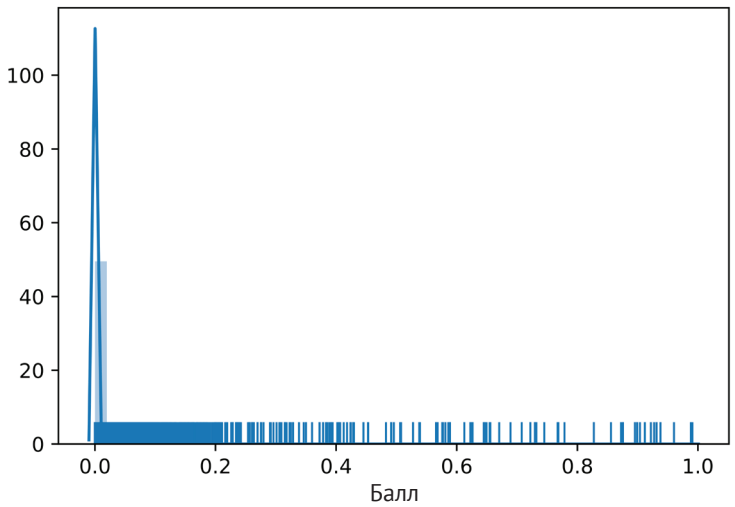


Рис. 11.10 ❖ График распределения баллов по прогнозам

Мы можем присоединить кадр данных с баллами к кадру данных со строками SMILES. Это дает нам возможность просматривать химическую структуру молекул с наибольшим количеством баллов:

```
combo_df = df.join(pred_df, how="outer")
combo_df.sort_values("Pos", inplace=True, ascending=False)
```

Как мы видели ранее, добавление столбца описания молекулы к информационному кадру позволяет нам взглянуть на химическую структуру молекул-победителей (рис. 11.11).

Судя по изображениям структур на рисунке, многие молекулы с наивысшим рейтингом структурно схожи между собой. Давайте посмотрим еще на несколько молекул (рис. 11.12). Значения под изображениями молекулярной структуры являются баллами, которые модель присвоила молекулам в качестве прогноза их активности.

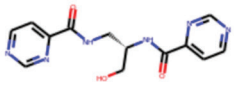
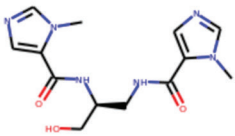
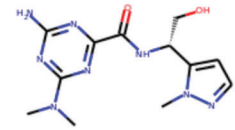
	SMILES	Name	Val	Neg	Pos	Mol
63669	O=C(NC[C@@H](CO)NC(=O)c1ccncn1)c1ccncn1	ZINC000681745616	0	0.438595	0.561404	
55121	Cn1cncc1C(=O)NC[C@@H](CO)NC(=O)c1cncn1C	ZINC000644062250	0	0.481628	0.518372	
38671	CN(C)c1nc(N)nc(C(=O)N[C@H](CO)c2ccnn2C)n1	ZINC000566403331	0	0.501487	0.498513	

Рис. 11.11 ❖ Химическая структура молекул с самыми высокими баллами

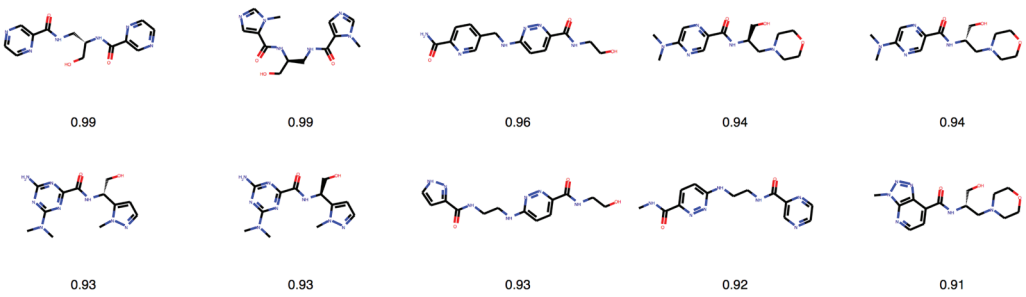


Рис. 11.12 ❖ Набор структур, получивших наивысшие баллы

Действительно, многие молекулы очень похожи друг на друга и могут оказаться лишними для нашего скрининга. Один из способов повышения эффективности – кластеризовать молекулы и проводить скрининг только молекул с наибольшим количеством баллов в каждом кластере. В RDKit реализован метод *кластеризации Бутины* (Darko Butina), один из наиболее широко используемых методов в хемоинформатике. В этом методе молекулы группируют на основе их химического сходства, вычисляемого путем сравнения битовых массивов 1 и 0 (битовых век-

торов) – то есть химических отпечатков, представляющих наличие или отсутствие определенных структур. Эти битовые векторы обычно сравниваются с использованием метрики, известной как *коэффициент Танимото* (Tanimoto):

$$Tanimoto = \frac{A \cap B}{A \cup B}.$$

Числитель уравнения – это *пересечение* или число битов, равное 1 в обоих битовых векторах A и B. Знаменатель – это *объединение* или число битов, равных 1 в векторе A или в векторе B. Коэффициент Танимото может находиться в диапазоне от 0 (молекулы не имеют общих паттернов атомов) до 1 (все паттерны, содержащиеся в молекуле A, также содержатся в молекуле B). В качестве примера мы можем рассмотреть битовые векторы, показанные на рис. 11.13. Пересечение двух векторов составляет 3 бита, а объединение – 5. Тогда коэффициент Танимото равен 3/5, или 0,6. Обратите внимание, что приведенный здесь пример был значительно упрощен для наглядности. На практике эти битовые векторы могут содержать сотни или даже тысячи битов.

```
A =    11011010
B =    11010000
A∩B =  11010000  Пересечение = 3

A =    11011010
B =    11010000
A∪B =  11011010  Объединение = 5
```

Рис. 11.13 ❖ Вычисление коэффициента Танимото

Для кластеризации набора молекул не требуется большой код. Единственным параметром, необходимым для кластеризации Бутины, является граничное значение рейтинга. Если сходство двух молекул по Танимото превышает пороговое значение, молекулы помещаются в один кластер. Если сходство меньше порогового значения, молекулы помещаются в разные кластеры:

```
def butina_cluster(mol_list, cutoff=0.35):
    fp_list = [
        rdmd.GetMorganFingerprintAsBitVect(m, 3, nBits=2048)
        for m in mol_list]
    dists = []
    nfps = len(fp_list)
for i in range(1, nfps):
    sims = DataStructs.BulkTanimotoSimilarity(
        fp_list[i], fp_list[:i])
    dists.extend([1 - x for x in sims])
mol_clusters = Butina.ClusterData(dists, nfps, cutoff, isDistData=True)
cluster_id_list = [0] * nfps
for idx, cluster in enumerate(mol_clusters, 1):
    for member in cluster:
        cluster_id_list[member] = idx
return cluster_id_list
```

Перед кластеризацией мы создадим новый кадр данных, содержащий только 100 молекул с наилучшими показателями. Поскольку набор `combo_df` уже отсортирован, достаточно воспользоваться функцией `head`, чтобы выбрать первые 100 строк в кадре данных:

```
best_100_df = combo_df.head(100).copy()
```

Затем мы можем создать новый столбец, содержащий идентификатор кластера для каждого соединения:

```
best_100_df["Cluster"] = butina_cluster(best_100_df.Mol)
best_100_df.head()
```

Как всегда, стоит посмотреть на данные и убедиться, что все работает. Теперь мы видим, что в дополнение к строке SMILES, имени молекулы и прогнозу рейтинга у нас также есть идентификатор кластера (рис. 11.14).

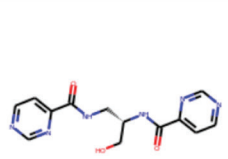
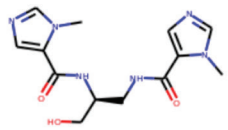
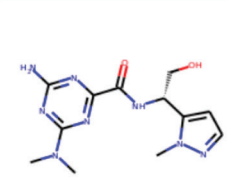
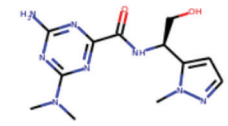
	SMILES	Name	Val	Neg	Pos	Mol	Cluster
63669	O=C(NC[C@@H](CO)NC(=O)c1ccncn1)c1ccncn1	ZINC000681745616	0	0.438595	0.561404		55
55121	Cn1cncc1C(=O)NC[C@@H](CO)NC(=O)c1cncn1C	ZINC000644062250	0	0.481628	0.518372		54
38671	CN(C)c1nc(N)nc(C(=O)N[C@H](CO)c2ccnn2C)n1	ZINC000566403331	0	0.501487	0.498513		26
38672	CN(C)c1nc(N)nc(C(=O)N[C@@H](CO)c2ccnn2C)n1	ZINC000566403338	0	0.501487	0.498513		26

Рис. 11.14 ❖ Первые несколько строк кластеризованного набора данных

Воспользовавшись функцией Pandas `unique()`, мы обнаружим, что у нас есть 55 уникальных кластеров:

```
len(best_100_df.Cluster.unique())
```

В конечном итоге мы хотели бы заказать синтез этих соединений и проверить их экспериментально. Для этого нам нужно сохранить файл CSV, в котором перечислены молекулы соединений, которые мы планируем заказать. Для исключения повторов и выбора одной молекулы на кластер можно воспользоваться функцией `drop_duplicates`. По умолчанию функция начинается с верхней части таблицы и удаляет строки со значениями, которые уже были обработаны:

```
best_cluster_rep_df = best_100_df.drop_duplicates("Cluster")
```

Просто чтобы убедиться, что эта операция работает, воспользуемся параметром `shape` и получим количество строк и столбцов в новом кадре данных:

```
best_cluster_rep_df.shape
```

Наконец, запишем файл CSV со списком молекул, подлежащих синтезу и экспериментальной проверке:

```
best_cluster_rep_df.to_CSV("best_cluster_represenatives.CSV")
```

Заключение

Итак, мы выполнили все шаги рабочего процесса виртуального скрининга на основе лигандов. Мы использовали глубокое обучение, чтобы построить классифицирующую модель, способную различать активные и неактивные молекулы. Процесс начался с оценки и балансировки обучающих данных. Мы добились того, чтобы молекулярный вес, LogP и распределения заряда были сбалансированы между наборами активных и «бесполезных» молекул. Как только внесены необходимые изменения в химическую структуру молекул-пустышек, мы готовы построить модель.

Первым шагом в построении модели было создание набора химических признаков для используемых молекул. Мы использовали функцию DeepChem `GraphConv` для создания набора соответствующих химических признаков. Эти признаки затем были применены для построения графовой сверточной модели, занятой в прогнозировании активности ряда коммерчески синтезируемых молекул. Чтобы исключить молекулы, которые могут создать проблемы в биологических экспериментах, мы применили набор вычислительных правил, закодированных как шаблоны SMARTS, и выявили молекулы, о которых ранее было известно, что они отрицательно влияют на эксперименты или создают определенные трудности.

Получив список молекул с высоким прогнозом активности, мы можем проверить эти молекулы в биологических экспериментах. Как правило, следующим шагом в нашем рабочем процессе будет получение образцов химических соединений для испытаний. Если молекулы поступают из корпоративной коллекции химических соединений, роботизированная система собирает образцы и подготавливает их для тестирования. Если соединения были приобретены из сторонних коммерческих источников, необходимо дополнительное взвешивание и разбавление буферной водой или другим растворителем.

Подготовив образцы, мы приступаем к биологическим экспериментам. Эти эксперименты могут охватывать широкий диапазон применений, от ингибирования роста бактерий до предотвращения пролиферации раковых клеток. Несмотря на то что тестирование молекул является последним этапом нашего виртуального скрининга, оно еще далеко от конца пути в проекте по открытию нового лекарства. После проведения первоначальных биологических экспериментов мы анализируем результаты скрининга. Если эксперименты подтвердили наши ожидания и мы нашли действительно активные молекулы, далее мы обычно находим и тестируем другие подобные молекулы, чтобы понять взаимосвязь между различными частями молекулы и биологической активностью, которую мы исследуем. Этот процесс оптимизации часто включает синтез и тестирование сотен или даже тысяч молекул, чтобы найти соединения, обладающие желаемой комбинацией безопасности и биологической активности.

Глава 12

Ожидания и перспективы

Науки о жизни развиваются с удивительной скоростью, возможно, быстрее, чем любая другая отрасль науки. То же самое можно сказать и о глубоком обучении: это одна из самых интересных и быстроразвивающихся областей информатики. Сочетание этих двух факторов способно разительно изменить мир и будет иметь далеко идущие последствия. Эффекты слияния двух отраслей уже начинают ощущаться, но это еще ничто по сравнению с тем, что может произойти в течение следующих нескольких десятилетий. Союз глубокого обучения с биологией может принести огромную пользу, но может и причинить столь же огромный вред.

В этой заключительной главе мы оставим в стороне механику обучения глубоких моделей и более подробно обсудим будущее отрасли. Где мы видим наибольший потенциал для решения важных проблем? Какие препятствия нужно преодолеть, чтобы это произошло? И каких рисков, связанных с этой работой, мы должны избегать?

Медицинская диагностика

Диагностика болезней была одним из первых практических применений глубокого обучения. Всего за последние несколько лет были опубликованы модели, которые сопоставимы по точности или даже превосходят опытных специалистов при диагностике многих важных заболеваний. В перечень достижений входит диагностика пневмонии, рака кожи, диабетической ретинопатии, возрастной дегенерации желтого пятна, аритмии сердца, рака молочной железы и многое другое. Ожидается, что этот список будет очень быстро расти.

Многие из этих диагностических моделей основаны на анализе изображений: рентгеновских снимков, МРТ, изображений с микроскопа и т. д. Это вполне объяснимо. Первые большие успехи глубокого обучения были достигнуты в области компьютерного зрения, и годы исследований позволили создать сложные архитектуры для анализа изображений. Применение этих архитектур к медицинским изображениям – очевидный низко висящий плод. Но не все применения основаны на изображениях. Для глубоких моделей подходят любые данные, представленные в числовой форме: электрокардиограммы, развернутые анализы крови, последовательности ДНК, *профили генной экспрессии* (gene expression profile), показатели жизнедеятельности и многое другое.

Во многих случаях самой сложной проблемой оказывается создание наборов данных, а не проектирование архитектур. Обучение глубокой модели требует много последовательных, четко размеченных данных. Если вы хотите диагности-

ровать рак по изображениям клеточной микроскопии, вам нужно много изображений тканей пациентов, как с раком, так и без него, с маркировкой, указывающей, кто есть кто. Если вы хотите диагностировать рак по экспрессии генов, вам нужно много помеченных профилей генной экспрессии. То же самое верно для каждой болезни, которую вы хотите диагностировать, и для каждого типа данных, из которого вы надеетесь извлекать диагноз.

В настоящее время многие из этих наборов данных не существуют. И даже когда соответствующие наборы данных существуют, они часто меньше, чем хотелось бы. Данные могут быть зашумленными, собранными из многих источников с систематическими различиями и ошибками. Многие из меток могут быть неточными. Некоторые данные существуют только в человекочитаемой форме, а не в форме, удобной для чтения машиной, например рукописный текст в свободной форме, записанный врачами в медицинские карты пациентов.

Прогресс в использовании глубокого обучения для медицинской диагностики принципиально зависит от появления лучших наборов данных. Иногда это означает объединение и обработку существующих данных. В других случаях приходится собирать новые данные с нуля, чтобы они подходили для машинного обучения. Последний подход часто дает лучшие результаты, но он также намного дороже.

К сожалению, создание массивных наборов данных может иметь катастрофические последствия для конфиденциальности пациентов. Медицинские записи содержат некоторые из наших самых важных и интимных данных. Хотите ли вы, чтобы о поставленном диагнозе узнал работодатель? А ваши соседи? Ваша страховая компания? А как насчет рекламодателей, которые мечтают о новой возможности продавать вам товары, связанные со здоровьем?

Проблемы конфиденциальности особенно важны при использовании генетических последовательностей, потому что они обладают уникальным свойством распределения между родственниками. Ваши родители, ваш ребенок, ваш брат или сестра – в каждом из них есть доля в 50 % вашей ДНК. Невозможно раскрыть генетическую последовательность одного человека, не предоставив при этом много информации обо всех его родственниках. Также невозможно анонимизировать эти данные. Ваша последовательность ДНК идентифицирует вас гораздо точнее, чем ваше имя или ваш отпечаток пальца. Пока что вопрос использования генетических данных без нарушения конфиденциальности остается большой проблемой.

Рассмотрим факторы, которые делают данные наиболее полезными для машинного обучения. Во-первых, разумеется, данных должно быть много. Забирайте себе столько данных, сколько можете получить. Данные должны быть незашумленными, подробными и точно маркированными. Данные должны быть легкодоступны. Многие исследователи захотят использовать ваши данные для обучения своих моделей. Данные одного вида должны быть легко сопоставимы с другими данными. Например, последовательности ДНК, профили экспрессии генов и история болезни полезны и по отдельности, но только представьте, сколько полезных выводов можно извлечь, если объединить эти разнотипные данные для одних и тех же пациентов!

Теперь рассмотрим факторы, которые делают данные наиболее подверженными злоупотреблениям. Нам даже не нужно перечислять их, потому что мы только что это сделали. Факторы, делающие данные полезными, в то же время облегчают

злоупотребления. Поиск компромисса между использованием генетических данных и сохранением тайны частной жизни остается одной из важнейших задач на ближайшие годы.

Персонализированная медицина

Следующим шагом после диагностики болезни является поиск эффективного лечения. Традиционно это работало по принципу «один размер подходит всем»: препарат рекомендуется при заболевании, если он помогает некоторой достаточно большой части пациентов с таким диагнозом, не вызывая при этом слишком много побочных эффектов. Ваш врач может спросить, есть ли у вас какие-либо известные аллергии или критические противопоказания, но это предел персонализации.

Такой подход игнорирует сложность человеческого организма. Каждый человек уникален. Препарат может быть эффективным для одних людей и бесполезным для других. Он может вызвать серьезные побочные эффекты у одних людей и не вызвать у других. У некоторых людей могут быть ферменты, очень быстро расщепляющие лекарство, и поэтому им нужна увеличенная доза, в то время как другие обойдутся гораздо меньшей дозой.

Диагнозы – это всего лишь очень грубые описания. Когда врач заявляет, что у пациента диабет или рак, это может означать много разных вещей. На самом деле каждый рак уникален, это различные наборы мутаций, которые заставляют клетки стать раковыми. Лечение, которое работает для одного, может не работать для другого.

Персонализированная медицина – это попытка выйти за рамки обобщающего подхода. Она пытается принять во внимание уникальную генетику и биохимию каждого пациента, чтобы выбрать лучшее лечение для конкретного человека, то есть наибольшую пользу с наименьшим количеством побочных эффектов. В принципе, это может привести к резкому улучшению качества медицинской помощи.

Если персонализированная медицина реализует свой потенциал, компьютеры будут в ней играть центральную роль. Чтобы предсказать, как возможный препарат будет взаимодействовать с уникальной биологией и состоянием конкретного пациента, необходимо проанализировать огромный объем данных, гораздо больше, чем может обработать человек. Глубокое обучение отлично справляется с такой проблемой.

Как мы обсуждали в главе 10, *интерпретируемость* и *объяснимость* имеют решающее значение для применения глубокого обучения в медицине. Когда компьютер выводит диагноз и рекомендует лечение, врачу нужен способ дважды проверить эти результаты и решить, доверять им или нет. Модель должна объяснить, почему она пришла к своему заключению, представив понятные и доступные для проверки доказательства.

К сожалению, объемы данных и сложность биологических систем в конечном итоге превзойдут способность любого человека понимать объяснения. Если модель «объяснит», что уникальная комбинация мутаций пациента с 17 генами сделает конкретное лекарство эффективным именно для этого пациента, ни один врач не сможет проверить истинность этого утверждения. Это создает практические, правовые и этические проблемы, которые необходимо будет решить. В каком случае врач может назначить лечение, не понимая, почему оно рекоменду-

ется? Когда ему следует игнорировать рекомендации компьютера и назначать что-то еще? В любом случае, кто несет ответственность, если назначенное лечение не сработает или проявятся опасные для жизни побочные эффекты?

Персонализированная медицина, вероятно, будет проходить через ряд этапов. Сначала компьютеры будут только помощниками врачей, помогая им лучше понимать данные. Со временем компьютеры далеко превзойдут людей при выборе методов лечения, поэтому для любого врача будет совершенно неэтично противоречить им. Но это займет много времени, и мы не обойдемся без продолжительного переходного периода. Во время этого переходного периода у врачей будет возникать искушение доверять компьютерным моделям, которым, возможно, еще нельзя доверять, и полагаться на их рекомендации больше, чем они заслуживают. Как человек, создающий эти модели, вы должны тщательно продумать, как они будут использоваться. Хорошо подумайте о том, какие результаты нужно получить и как эти результаты представить пользователю, чтобы минимизировать вероятность того, что кто-то их неправильно поймет или придаст слишком большое значение ненадежному результату.

Фармацевтические исследования

Процесс разработки нового препарата чрезвычайно сложен и занимает много времени. Глубокое обучение способно решить многие проблемы фармацевтов, о чем мы уже говорили в этой книге.

Это также очень дорогой процесс. Недавнее исследование показало, что фармацевтические компании тратят в среднем 2,6 млрд долл. на исследования и разработки для каждого одобренного препарата. Конечно, это не означает, что разработка одного препарата обходится в миллиарды долларов. Это означает, что большинство кандидатов в лекарства терпит неудачу. Прежде чем получить один одобренный препарат, компания тратит деньги на разработку многих других соединений, но в итоге отказывается от них.

Хотелось бы сказать, что глубокое обучение собирается охватить и решить все проблемы, но это маловероятно. Увы, фармацевтика слишком сложна. Когда лекарство попадает в ваше тело, оно вступает в контакт с сотнями тысяч других молекул. Вам нужно, чтобы оно взаимодействовало с правильными молекулами и только правильным способом, чтобы получить желаемый эффект, но при этом не взаимодействовало с другими молекулами, вызывая отравление или иные нежелательные побочные эффекты. Лекарство должно быть хорошо растворимым для попадания в кровь, а в некоторых случаях должно преодолевать *гематоэнцефалический барьер*. Вдобавок, попав в организм, многие лекарства подвергаются химическим реакциям, по-разному меняющим их свойства. Вы должны учитывать не только эффекты оригинального препарата, но и эффекты всех продуктов, произведенных из него! Наконец, добавьте сюда требования, что препарат должен быть недорогим в производстве, иметь длительный срок хранения, быть простым в применении и т. д.

Разработка лекарств невероятно сложна. В ней слишком много разных факторов, чтобы оптимизировать их все сразу. Модель глубокого обучения способна помочь в решении отдельных проблем, но каждая из них представляет лишь крошечную часть процесса.

Однако можно взглянуть на ситуацию и по-другому. Невероятная стоимость разработки лекарств означает, что даже небольшие улучшения могут принести большую отдачу. Предположим, что 5 % из 2,6 млрд долл. – это 130 млн долл. Если глубокое обучение может снизить стоимость разработки лекарств на 5 %, это быстро приведет к экономии миллиардов долларов.

Процесс разработки лекарств можно условно изобразить в виде воронки (рис. 12.1). Самые ранние стадии могут включать скрининг десятков или сотен тысяч соединений в поиске необходимых свойств. Хотя количество соединений огромно, стоимость отдельного анализа очень мала. Затем несколько сотен наиболее перспективных соединений выбирают для гораздо более дорогих доклинических исследований с участием животных или культивируемых клеток. Из них, возможно, 10 или меньше могут перейти к клиническим испытаниям на людях. И только один из них, если нам повезет, в конечном итоге выйдет на рынок как одобренный препарат. На каждом этапе количество соединений-кандидатов значительно сокращается, но стоимость каждого эксперимента растет еще быстрее, поэтому основная часть расходов приходится на более поздние этапы.

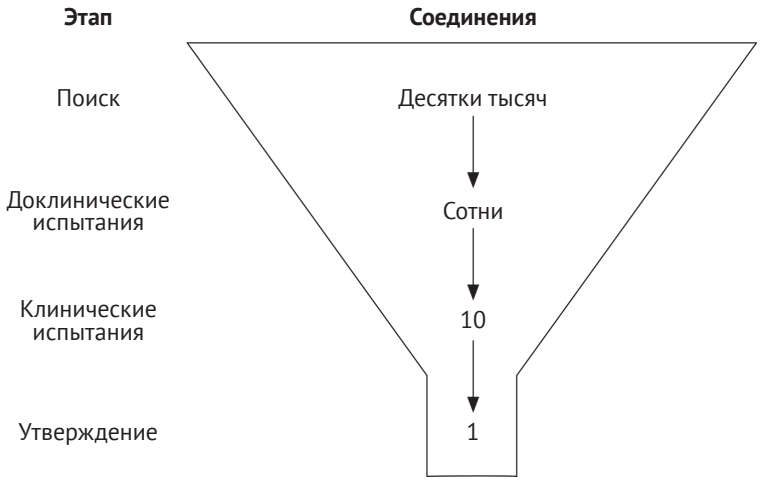


Рис. 12.1 ❖ Воронка разработки лекарств

Таким образом, хорошая стратегия снижения стоимости разработки лекарств выглядит очень просто: «*Отказаться как можно раньше*». Если соединение в конечном итоге будет отклонено, постарайтесь отфильтровать его на ранних стадиях процесса разработки до того, как сотни миллионов долларов будут потрачены впустую на клинических испытаниях. Если глубокая модель сможет более точно предсказывать на раннем этапе, какие соединения следует отбросить, не доводя до испытаний, экономия средств будет огромной.

Биологические исследования

В дополнение к своим медицинским применениям глубокое обучение имеет большой потенциал в фундаментальных исследованиях. Современные экспериментальные методы, как правило, производят обширные наборы данных, тысячи или миллионы чисел на эксперимент. Осмысление этих данных – огромная проблема. Глубокое обучение является мощным инструментом для анализа экспериментальных данных и выявления закономерностей в них. Мы показали это на примерах генетических данных и глубокой микроскопии.

Другая интересная перспектива заключается в том, что нейронные сети в своем исходном виде могут служить моделями биологических систем. Наиболее очевидно применение этой идеи в *нейробиологии*. В конце концов, устройство нейронных сетей непосредственно копирует нейронные связи в мозге. Как далеко заходит сходство? Если вы обучаете нейронную сеть для решения задачи, делает ли она это так же, как мозг?

По крайней мере, в некоторых случаях ответ – да! Это было доказано для нескольких различных функций мозга, включая обработку зрительных[1], слуховых[2] и двигательных ощущений. В каждом случае нейронную сеть обучали для выполнения задачи. Затем ее сравнивали с соответствующей областью мозга и каждый раз обнаруживали, что нейросеть хорошо соответствует поведению мозга. Например, определенные слои в сети могут использоваться для точного прогнозирования поведения определенных областей в зрительной или слуховой коре.

Это весьма примечательно. Модели не были специально разработаны для копирования какой-либо конкретной области мозга. В каждом случае исследователи просто создавали общую модель и обучали ее методом оптимизации градиентным спуском для выполнения некоторой функции – и решение, найденное оптимизатором, оказалось практически таким же, как и решение, найденное миллионами лет эволюции. Фактически нейронная сеть оказалась ближе к системе мозга, чем другие модели, специально разработанные для ее имитации!

Чтобы развиваться в этом направлении дальше, нам, вероятно, потребуется разработать совершенно новые архитектуры. Сверточные сети были во многом скопированы со зрительной коры, поэтому сверточная нейронная сеть является хорошей моделью кортекса. Но, вероятно, есть и другие области мозга, которые работают совершенно иначе. Возможно, это приведет к продуктивному обмену идеями между нейробиологией и глубоким обучением: нейробиологи предложат

---

[1] *Yamins Daniel L. K., et al*. Performance-Optimized Hierarchical Models Predict Neural Responses in Higher Visual Cortex // Proceedings of the National Academy of Sciences 111:8619–8624. 2014. URL: **https://doi.org/10.1073/pnas.1403112111**.

[2] *Kell Alexander J. E., et al*. A Task-Optimized Neural Network Replicates Human Auditory Behavior, Predicts Brain Responses, and Reveals a Cortical Processing Hierarchy // Neuron 98:630–644. 2018. URL: **https://doi.org/10.1016/j.neuron.2018.03.044**.

полезные новые архитектуры для глубокого обучения, и эти архитектуры, в свою очередь, послужат моделями для лучшего понимания мозга.

Разумеется, в биологии существуют и другие сложные системы. Как насчет иммунной системы или генетического механизма? Их можно рассматривать как «сети» с огромным количеством узлов, отправляющих информацию друг другу. Можно ли использовать глубокие модели для представления этих систем и лучшего понимания их работы? На этот вопрос пока нет ответа.

Заключение

Глубокое обучение – это мощный и быстро развивающийся инструмент. Если вы работаете в области наук о жизни, не забывайте о глубоком обучении, потому что оно способно радикально изменить вашу сферу деятельности.

Точно так же, если вы работаете в области глубокого обучения, науки о жизни являются невероятно важным объектом исследований, который заслуживает вашего внимания. Они предлагают комбинацию огромных наборов данных, сложных систем, которые трудно описать традиционными методами, и проблем, напрямую влияющих на благосостояние людей.

С какой бы стороны вы ни подходили к глубокому обучению, мы надеемся, что эта книга даст вам основу, опираясь на которую, вы внесете свой вклад в применение глубокого обучения. Мы застали замечательный момент в истории, когда много новых технологий соединяется вместе, чтобы изменить мир. Нам всем повезло быть частью этого процесса.

Колофон

Животное на обложке книги «Глубокое обучение и науки о жизни» – это самец джунглевой курицы Соннерата (Gallus sonneratii), также известной как серая джунглевая курица. Название вида sonneratii – дань уважения французскому натуралисту и исследователю Пьеру Соннера́[1] (Pierre Sonnerat). Курица Соннерата родом из южной и западной Индии, которую исследователь посещал несколько раз в период с 1774 по 1781 год. Естественная среда обитания птиц – лесной подлесок и бамбуковые заросли, но они хорошо проживают в самых разных условиях, от леса до тропиков и равнин.

Оперение куриц Соннерата покрыто белыми и коричневыми пятнами. На кончиках крыльев и хвостов у них черные перья с оттенком синего. Самцы и самки куриц Соннерата сильно различаются по многим параметрам. Петухи достигают около 75 см в длину, в то время как куры достигают всего около 40 см. Петухи ярче, чем курицы с блестящими хвостами, золотистыми крапинками и красноватым гребнем, ногами и бородкой. Ножки куриц желтые, а перья тускло-коричневые.

Цыплята рождаются с окраской от бледно-коричневой до бежевой. Куры обычно откладывают от 4 до 7 яиц в период с февраля по май. Они откладывают яйца в гнездах на земле, устланных травой и ветками, и высиживают в одиночку, без помощи петуха.

Курицы Соннерата являются предками одомашненных кур. Они могут скрещиваться с обыкновенными домашними курами и красными лесными курицами (оба Gallus gallus), создавая множество гибридов. Одомашненных куриц Соннерата содержат в загонах со сплошными стенами, потому что они своенравны и свободолюбивы. Пятнистые коричневые и белые перья обычно используются рыбаками для изготовления мушек.

Многие из животных на обложках O'Reilly находятся под угрозой исчезновения; все они важны для мира. Чтобы узнать больше о том, как вы можете помочь, посетите сайт animals.oreilly.com.

Иллюстрация на обложке выполнена Карен Монтгомери (Karen Montgomery) на основе черно-белой гравюры из книги «Иллюстрированное естествознание» Вуда (Wood's Illustrated Natural History).

---

[1] В России принято говорить «курица Соннерата». – *Прим. перев.*

Предметный указатель

C
CellProfiler, инструмент анализа клеток, 100
ChEMBL, база данных молекул, 167
CNN, convolutional neural network, 30

D
DeepChem, библиотека, 16
DeepPatient, система анализа ЭМК, 127
DUD-E, база данных соединений, 167

E
ECFP4, алгоритм фичеризации, 56
ECFP, extended connectivity fingerprints, 56

F, G, I, J, K
FHIR, формат обмена данными, 127
GRU, gated-recurrent unit, 31
ICD-10, классификатор заболеваний, 126
JUND, фактор транскрипции, 90
Jupyter Notebook, онлайн сервис, 78
Kaggle DR, набор данных, 136

M
MLP, multilayer perceptron, 22
MNIST, 42
MoleculeNet, коллекция наборов данных, 60
MUV, набор данных, 146

P
PDBBind, набор данных, 65
PDB, файл, 71

R
RDKit, пакет хеминформатики, 56
ReLU, rectified linear unit, 23
RNN, recurrent neural network, 31
ROC-AUC, 40

S, T
SAR, structure-activity relationship, 15
SGD, stochastic gradient descent, 26
SMARTS, 61
SMILES, 55
TenzorFlow, 33

А
Автокодер шумоподавляющий, 127
Автоэнкодер, 141
  вариационный, 143
  декодер, 142
  кодировщик, 142
  скрытое пространство, 142
Алгоритм оптимизации
  Adam, 26
  RMSProp, 26
Аминокислота, 67, 86
  боковые цепи, 67
  остаток, 78
Анализ ферментативный, 36
Антиген, 83
  участки связывания, 83
Антитело, 83
Аффинность связывания, 76

Б
Бактерии
  грамотрицательные, 109
  грамположительные, 109
Банк белковых структур, 67
Белок, 65, 68
  объемный снимок, 65
  терапевтически релевантный, 67
  частично неупорядоченный, 69
  шапероны, 89
Биологическая мишень, 36
Биофизика, 16
Ближнее поле, 105
  нераспространяющиеся волны, 108

В
Вектор, 19
Вектор химических дескрипторов, 48
Взаимосвязи структурно-функциональные, химическая информатика.
См. Хемоинформатика
Визуальная диагностика, 16
Виртуальный скрининг, 18
Вокселизация, 79
Воксель, 79
Выборка данных, 24

Выделение молекулярных признаков. *См.* Фичеризация молекулярная
Выпадение. *См.* Исключение
Вычисляемые молекулярные свойства, 149

Г

Генетика, 85
Геном, 85
Геномика, 85
Гиперболический тангенс, 23
Гиперпараметры, 15, 29
  оптимизация, 29
Гистология, 133
Гистон, 87
Глазное дно, 135
Глубокое обучение, 15
Гомолог, 69
Градиентный спуск, 25
  стохастический, 26
Граф, 52
  двухмерный, 48
  молекулярный, 52
  ребра, 52
  узлы, 52

Д

Данные эталонные экспериментальные, 118
Диабетическая ретинопатия, 135
Дискретизация
  повышение, 119
  снижение, 119
Дискриминатор, 143
Длина волны де Бройля, 105
ДНК
  ген, 87
  метилирование, 88
  основание, 85
    кодон, 86
  сайт связывания, мотив, 90
  хромосома, 87
Дополнение данных, 137
Дополняющие изображения, 137
Доступность данных, 152

Е, И

Евклидово расстояние, 25
Исключение, 28

К

Каппа Коэна, 138
Картирование значимости, 155
Кластеризация
  Бутины, 183
  записей, 127
Клеточная
  линия, 114
  сегментация, 117
Количественная оценка сходства, 149
Кольцо
  ароматическое, 73
  гетерогенное, 74
Коробочная диаграмма, 168
Коэффициент
  корреляции Мэтьюса, 173
  корреляции Пирсона, 59
  Танимото, 184
КТ, компьютерная томография, 129

Л

Лиганд, 65
Линейная регрессия, 15
Логарифм коэффициента распределения, 57
Логистическая сигмоида, 23
Логический движок, 124

М

Макромолекулярный комплекс, 72
Маммограмма, 132
Маска сегментации, 117
Масс-спектроскопия, 49
Машинное обучение, 15
Мезосома, 113
Метаматериалы, 108
Метка, 38
Метрика, 40
Микроскоп
  атомно-силовой, 107
    кантилевер, 107
  конфокальный, 102
  оптический, 101
    простой, 101
    составной, 101
  просвечивающий электронный, 106
  флуоресцентный, 111
  фокальная плоскость, 102
Микроскопия, 99, 101
  автоматизированная скоростная, 100
  ближнего поля, 108
  детерминированное сверхразрешение, 108
  дифракционный предел, 104
  криоэлектронная, 66
  микротом, 111
  окрашивание
    избирательное, 109

по Граму, 109
оптическое секционирование, 102
препарат, 101
пробоподготовка, 109
сверхвысокого разрешения, 107
стохастическая, 108
фототоксичность, 104
функциональная, 107
числовая апертура, 104
Микроэксперимент, 14
МКБ-10. *См.* ICD-10
Многозадачная проблема, 39
Многоклассовая классификация, 138
Моделирование
гомологичное, 69
физическое, 69
Модель, 20
генеративная, 17, 141
глубокая, 23
дерево решений, 81
дискриминантная, 17
интерпретируемость, 154
линейная, 20
неглубокая, 23
область применимости, 153
объяснимая, 165
параметры, 20
преобразователь изображения, 118
прогностическая, 15
случайный лес, 80
точность, 47
точность предсказаний, 18
Молекула, 48
R-форма, 54
S-форма, 54
конформация, 53
линкер, 53
токсичность, 16
формальный заряд, 168
хиральность, 54
аксиальная, 54
МРТ, магниторезонансная томография, 129, 133
Муковисцидоз, 15

Н

Набор данных
категорийный, 45
обучающий, 15, 20
оценочный, 26
проверочный, 15, 29
Нейронная сеть, 23
глубокая, 13
рекуррентная, 31
сверточная, 30

О

Обучение
контролируемое, 127
неконтролируемое, 127
эпоха, 40
Орбитальная функция, 48
Органеллы, 99
Ошибка среднеквадратичная, 161

П

Пакет, 26
Пептид, 68
Перекрестная энтропия, 25
Переобучение, 27
Персептрон, 20
многослойный, 22
глубина, 23
ширина слоя, 23
скрытый слой, 22
стек слоев, 22
Позиционная весовая матрица, 90
Полимер, 85
Полифармакология, 64
Полуацеталь, 151
Полярная площадь поверхности, 57
Предиктор, 48
Предсказательное моделирование, 18
Преобразователь, 39
Признак, 34
Признаковое описание, 36
Проба, 18
Проспективные испытания, 131
Прямое унитарное кодирование, 45
Прямой унитарный код, 45

Р

Радиография, 131
Радиология, 129
Распознавание
изображений, 14
речи, 14
Растворимость, 59
Регуляризация, 27
Рекомендательные системы, 14
Рентгеновская кристаллография, 66
рентгеновский снимок, 129
Рентгенография, 131
Рецепторное поле, 30
Рибосомы, 88

РНК
индуцированный комплекс молчания, 96
интерференция, 96
интерферирующая малая, 96
информационная. *См.* РНК: матричная
короткая интерферирующая, 88
матричная, 87
микрофрагмент, 88
рибозимы, 89
рибосвитчи, 89
сплайсинг, 88
транспортная, 88
экзоны, 88

С

Свертка графовая, 48
Сверточное ядро, 31
Свободная энергия, 76
Связываемость, 64
Связь
белок–лиганд, 65
водородная, 51, 71, 72
дополненные отпечатки, 56
ковалентная, 51
нековалентная, 51
пи-связь, 74
пи-стекинг, 51, 73
расстояние отсечки, 73
соляной мостик, 51, 71, 73
химическая, 49
Секвенирование генов, 14
Сеть
байесовская, 125
генеративная состязательная, 141
графовая сверточная, 58
Сигнальная трансдукция, 70
Синтетические выборки, 143
Система
диагностическая, 123
экспертная, 124
Скорость обучения, 25
Скрининг
виртуальный, 166
высокопроизводительный, 166
на основе лигандов, 166
на основе структуры, 166
Скрипичный график, 168
Слой
исключения, 92
полносвязный, 91
полностью связанный, 30
сверточный, 30
Случайное начальное число, 34
Смесь рацемическая, 55
Спиросоединение, 54
Степень гибридизации орбиталей, 58

Т

Терапевтический отклик, 70
Терапия
антиретровирусная, 13
программно-ориентированная, 135
Тератогенность, 55
Тиамин, 56
Токены, 147

У

УЗИ, ультразвуковое исследование, 129
Универсальный аппроксиматор, 24

Ф

Фактор транскрипции, 87
Фиксация, 110
агент фиксирующий, 110
перфузия, 110
погружением, 110
тепловая, 110
Фичеризация, 36, 56
атомарная, 71
воксельная, 81
координатная, 71
молекулярная, 48
Флуоресценция, 111
Флуорофор, 105, 112
маркировка, 112
Функция
активации, 22
линейная выпрямленная, 23
логистическая сигмоидная, 92
логит, 92
потерь, 25
потерь кросс-энтропийная, 91

Х, Ш

Хемоинформатика, 15, 55
Хроматин, 93
Шаблон данных, 14

Э

Электронная медицинская карта, 126
Электростатическое распределение, 48
Эпистемическая неопределенность, 163
Эукариоты, 87

Я

Ядерный магнитный резонанс, 66
Языковой перевод, 14

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Бхарат Рамсундар, Питер Истман,
Патрик Уолтерс и Виджай Панде

Глубокое обучение в биологии и медицине

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Яценков В. С.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70 × 100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 16,25. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**