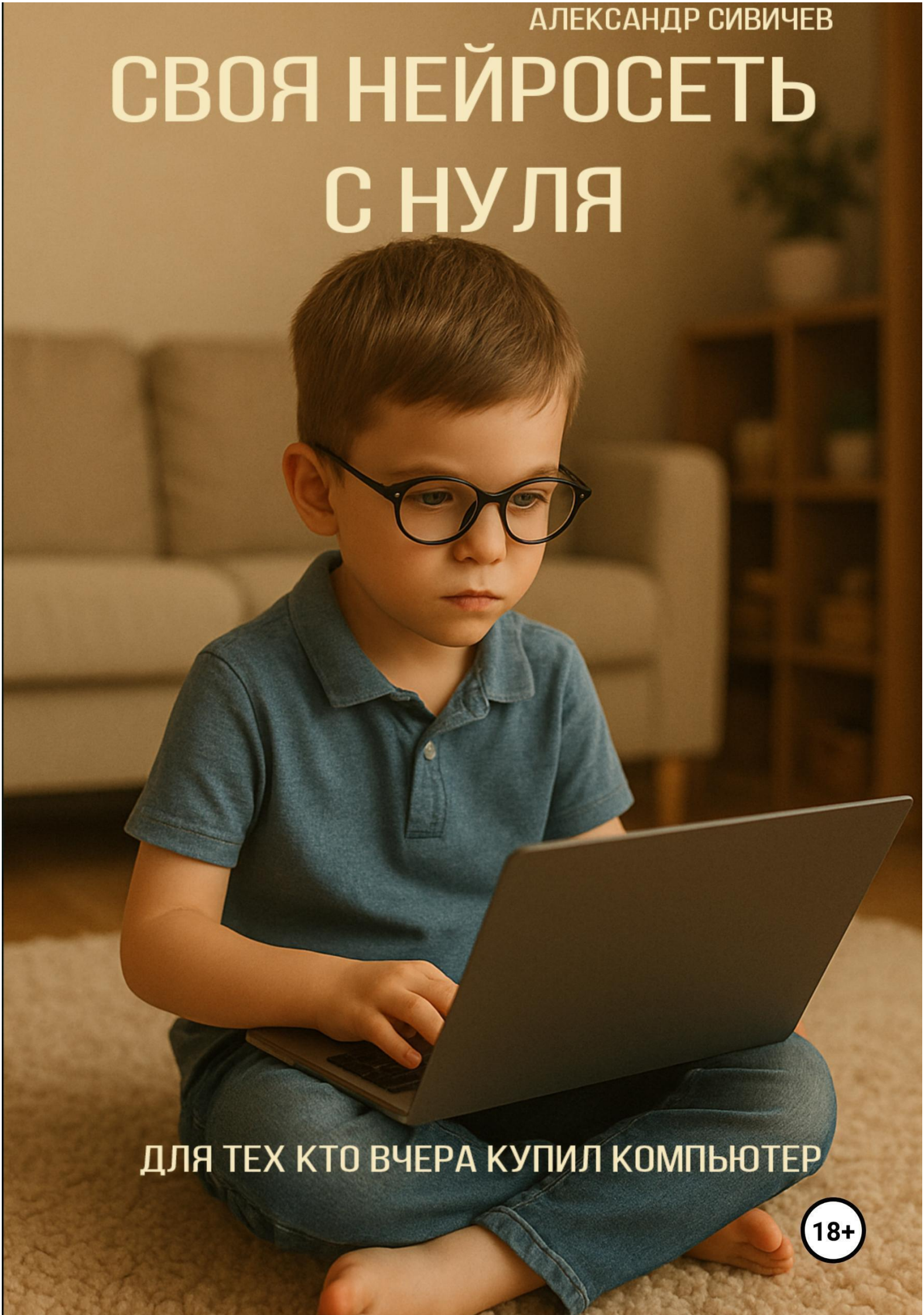
АЛЕКСАНДР СИВИЧЕВ

СВОЯ НЕЙРОСЕТЬ С НУЛЯ



ДЛЯ ТЕХ КТО ВЧЕРА КУПИЛ КОМПЬЮТЕР

18+

Александр Сивичев

Самоучитель: своя нейросеть с нуля. Для тех, кто вчера купил компьютер

«Автор»
2025

Сивичев А.

Самоучитель: своя нейросеть с нуля. Для тех, кто вчера купил компьютер / А. Сивичев — «Автор», 2025

Самоучитель: своя нейросеть с нуля. Для тех, кто вчера купил компьютер — практическое руководство по созданию и использованию нейросетей, рассчитанное на читателя без технического образования. Книга пошагово проведёт от простейшего перцептрона до собственной модели, готовой к интеграции в веб-сервис или Telegram-бота. Вы узнаете, как работают современные архитектуры, освоите библиотеки PyTorch, TensorFlow и Hugging Face, научитесь обрабатывать текст, изображения, звук и создавать мультимодальные решения. Особое внимание уделено практическим проектам, оптимизации моделей и деплою на ПК, сервере и в облаке. Все примеры снабжены готовым кодом, который можно запустить и адаптировать под свои задачи.

Александр Сивичев
Самоучитель: своя нейросеть с нуля. Для тех, кто вчера купил компьютер

Введеиие или зачем тебе вообще своя нейросеть

Если ты держишь в руках эту книгу, скорее всего, ты хочешь разобраться в том, как работают нейросети, и, возможно, создать свою. Не просто "поиграться", не просто нажать на кнопку, а понять суть – и собрать инструмент, который работает под твои задачи. Это правильная цель.

Сегодня нейросети повсюду – в поиске, в рекламе, в банковских решениях, в голосовых помощниках и в камерах смартфонов. Они пишут тексты, рисуют картинки, подбирают музыку и советуют фильмы. Но вот парадокс: чем больше таких сервисов становится «доступными и бесплатными», тем больше пользователей начинают задаваться вопросом: а кто в этой цепочке – продукт?

Опыт общения с крупными нейросетевыми системами показывает: если продукт бесплатен – продуктом становишься ты сам. Ты, твои запросы, твой стиль общения, твои данные. Всё это анализируется, обрабатывается, запоминается – но не ради тебя. Не чтобы ты в следующий раз получил «лучший ответ». А чтобы система стала умнее, быстрее, точнее. Ты – обучающая выборка.

Иногда эти сервисы и вовсе устроены так, что никаких «объёмов пользовательских данных» они не хранят: после сессии твои данные выгружаются, освобождая память. В лучшем случае они попадают в обезличенный журнал, в худшем – просто исчезают. В таких условиях бессмысленно говорить о «настройке» под тебя, об «обучении на твоих задачах». Это не инструмент – это стенд для демонстрации.

Да, это может быть забавно. Да, иногда получается красиво. Но в долгосрочной перспективе это тупиковый путь: ты зависишь от интерфейса, не контролируешь модель, не понимаешь, как она принимает решения – и не можешь встроить её в свои процессы. У тебя нет инструмента – только интерфейс к чужому серверу.

Поэтому мы пойдём другим путём

В этой книге мы не будем изучать математику ради математики. Мы не будем писать километры кода, чтобы «почувствовать дух open source». Мы пойдём по прямой линии: от понимания к реализации, от простого к работающему.

Ты узнаешь:

* как устроен искусственный нейрон и как из них собирается сеть;
* как обрабатывать данные, чтобы с ними могла работать модель;
* как обучить свою сеть и проверить, действительно ли она «поняла» задачу;
* как использовать готовые модели и адаптировать их под себя;
* и как в итоге встроить свою нейросеть в реальное приложение – хоть на сайте, хоть в CRM, хоть в чат-боте.

И ты сделаешь это на своём компьютере. Даже если у тебя нет видеокарты. Даже если ты только вчера узнал, что Python – это язык программирования, а не змея.

Наша цель – не поиграть в нейросеть, а создать свой инструмент.

И ты с этим справишься. Начнём.

Кто может освоить нейросети

– даже если ты гуманитарий

Многие уверены, что нейросети – это «не для них». Что нужно иметь математическое образование, знать линейную алгебру, разбираться в производных и строить графы с трёхмерными функциями. На самом деле – всё наоборот.

Современные инструменты машинного обучения устроены так, чтобы прятать сложность. Ты не обязан понимать, как устроен внутренний градиентный буфер в PyTorch или каким методом считается кросс-энтропия. Твоя задача – понять общую структуру, принципы работы, где что подключается, и как это применить к своей задаче.

Если ты умеешь логически мыслить, умеешь формулировать проблему и разбираться в чужом коде – ты справишься. Если ты гуманитарий – у тебя даже есть преимущество: ты умеешь работать со смыслами, структурами, языком. Всё это становится крайне полезным, когда ты начинаешь обучать модель распознавать тексты, классифицировать поведение клиентов или вести разговор.

Это не про диплом. Это про настройку мышления.

И если ты читаешь это – ты уже на правильном пути.

Что получится в итоге:

чат-бот, классификатор, мини-ИИ

Мы не будем теоретизировать. Эта книга построена так, чтобы на каждом этапе у тебя появлялся работающий результат. Да, сначала это будет простейший перцептрон. Потом – чуть более серьёзная модель. Но уже к середине книги ты сможешь:

* обучить свою нейросеть на конкретной задаче: например, распознать, негативный ли отзыв оставил клиент;

* настроить классификатор, который делит входящие обращения на важные и второстепенные;

* создать мини-ИИ, который отвечает на вопросы, ведёт диалог или выполняет команды;

* встроить свою модель в интерфейс – чтобы можно было не только смотреть на код, но и использовать результат как полноценный инструмент.

Ты научишься не просто запускать нейросеть.

Ты научишься работать с ней как с инструментом, который принадлежит тебе.

Глава 1. Что такое нейросеть, и зачем она тебе

Примеры из жизни:

как работает автодополнение, фильтры, рекомендации

Ты уже сталкивался с нейросетями, даже если не знал об этом. Каждый раз, когда ты набираешь сообщение в мессенджере, и система подсказывает тебе следующее слово – это результат работы языковой модели. Когда ты открываешь YouTube или маркетплейс и тебе показывают «рекомендации специально для вас» – это работает система на основе поведения миллионов других пользователей, и она решает, что тебе предложить.

Фильтры в камере, распознавание лиц, автофокус, сортировка почты, реклама, поиск – всё это работа моделей.

Иногда они кажутся «умными», иногда – нелепыми. Почему? Потому что они обучены не на твоём опыте, а на чужом. Именно поэтому ты читаешь эту книгу: чтобы создать свою модель, заточенную под свою задачу, свой контекст, свои данные.

Нейросети в повседневной жизни выглядят как магия, но это просто алгоритмы.

Разница между «поиграться в нейросеть» и реально внедрить её в свою работу – это и есть то, чему ты здесь научишься.

Интуитивное объяснение:

аналогия с мозгом (но без мистики)

Нейросеть называется так не потому, что она «разумна», а потому что вдохновлена устройством мозга. В самом упрощённом виде можно сказать: нейросеть – это система, в кото-

рой отдельные «нейроны» получают вход, передают сигнал дальше, и в результате формируется выход.

Вот как это работает:

* Каждый нейрон – это мини-функция, которая получает числа на вход, умножает их на какие-то веса, применяет некую «функцию активации» – и передаёт результат дальше.

* Нейроны собираются в слои. Первый слой получает данные (например, пиксели изображения, или цифры из таблицы), последний – выдаёт результат (например, «это кошка» или «этот клиент не вернёт долг»).

* Между ними могут быть десятки или сотни скрытых слоёв, которые трансформируют данные, извлекают признаки, фильтруют шум.

Но и это упрощение. Главное, что тебе нужно понять:

Нейросеть – это способ научить машину принимать решения не по жёстким правилам, а по примерам.

Ты не пишешь правила. Ты показываешь примеры. А система сама настраивает свои веса и связи так, чтобы результат был как можно ближе к нужному.

Это и есть суть: не кодировать поведение напрямую, а обучать систему на данных.

Ты не говоришь: «если клиент позвонил три раза – это угроза».

Ты просто показываешь ей сотни похожих случаев – и она сама найдёт закономерности.

Если ты гуманитарий, представь, что нейросеть – это как стажёр, которому ты показываешь пачку писем и говоришь: «Вот смотри, вот это – жалобы, а вот это – благодарности. Учись отличать».

Он делает ошибки, ты их исправляешь – и со временем он становится всё точнее. Только этот стажёр работает быстрее, не забывает, и может за день обработать тысячу писем.

Глава 1. Что такое нейросеть, и зачем она тебе

Примеры из жизни:

как работает автодополнение, фильтры, рекомендации

Ты уже сталкивался с нейросетями, даже если не знал об этом. Каждый раз, когда ты набираешь сообщение в мессенджере, и система подсказывает тебе следующее слово – это результат работы языковой модели. Когда ты открываешь YouTube или маркетплейс и тебе показывают «рекомендации специально для вас» – это работает система на основе поведения миллионов других пользователей, и она решает, что тебе предложить.

Фильтры в камере, распознавание лиц, автофокус, сортировка почты, реклама, поиск – всё это работа моделей.

Иногда они кажутся «умными», иногда – нелепыми. Почему? Потому что они обучены не на твоём опыте, а на чужом. Именно поэтому ты читаешь эту книгу: чтобы создать свою модель, заточенную под свою задачу, свой контекст, свои данные.

Нейросети в повседневной жизни выглядят как магия, но это просто алгоритмы.

Разница между «поиграться в нейросеть» и реально внедрить её в свою работу – это и есть то, чему ты здесь научишься.

Интуитивное объяснение:

аналогия с мозгом (но без мистики)

Нейросеть называется так не потому, что она «разумна», а потому что вдохновлена устройством мозга. В самом упрощённом виде можно сказать: нейросеть – это система, в которой отдельные «нейроны» получают вход, передают сигнал дальше, и в результате формируется выход.

Вот как это работает:

* Каждый нейрон – это мини-функция, которая получает числа на вход, умножает их на какие-то веса, применяет некую «функцию активации» – и передаёт результат дальше.

* Нейроны собираются в слои. Первый слой получает данные (например, пиксели изображения, или цифры из таблицы), последний – выдаёт результат (например, «это кошка» или «этот клиент не вернёт долг»).

* Между ними могут быть десятки или сотни скрытых слоёв, которые трансформируют данные, извлекают признаки, фильтруют шум.

Но и это упрощение. Главное, что тебе нужно понять:

Нейросеть – это способ научить машину принимать решения не по жёстким правилам, а по примерам.

Ты не пишешь правила. Ты показываешь примеры. А система сама настраивает свои веса и связи так, чтобы результат был как можно ближе к нужному.

Это и есть суть: не кодировать поведение напрямую, а обучать систему на данных.

Ты не говоришь: «если клиент позвонил три раза – это угроза».

Ты просто показываешь ей сотни похожих случаев – и она сама найдёт закономерности.

Если ты гуманитарий, представь, что нейросеть – это как стажёр, которому ты показываешь пачку писем и говоришь: «Вот смотри, вот это – жалобы, а вот это – благодарности. Учись отличать».

Он делает ошибки, ты их исправляешь – и со временем он становится всё точнее. Только этот стажёр работает быстрее, не забывает, и может за день обработать тысячу писем.

Почему ИИ – это не магия и не дорого

Пока нейросети оставались прерогативой университетов, лабораторий и IT-гигантов, вокруг них вырастало ощущение таинственности. Мол, это что-то вроде квантовой физики: дорого, сложно, непонятно. Но ситуация изменилась. Сегодня, чтобы обучить свою нейросеть, тебе не нужен суперкомпьютер. Достаточно обычного ноутбука – или даже арендованного сервера за 5 долларов в месяц.

Что стало проще:

* Открытые библиотеки типа PyTorch, TensorFlow, scikit-learn сделали интерфейс понятным даже для новичков.

* Готовые датасеты, модели и туториалы лежат в свободном доступе – бери и используй.

* Визуальные инструменты вроде Gradio позволяют превращать модель в полноценное приложение за пару строк кода.

А главное – ты больше не один. Вокруг нейросетей сформировалось огромное сообщество. На форумах, в репозиториях GitHub, на StackOverflow люди делятся кодом, задают вопросы и помогают друг другу. Это не тайное знание. Это уже обычный цифровой навык – как когда-то стали HTML, Excel и Photoshop.

А что насчёт магии?

Да, иногда результат поражает. Особенно когда модель «угадывает» намерения пользователя или генерирует нечто неожиданное. Но под этой магией всегда лежит статистика, математика и примеры. Мы просто привыкли называть магией то, что работает, но непонятно как.

Хорошая новость: после этой книги тебе будет понятно, как.

Архитектура на пальцах: нейрон, слой, веса, активация

Чтобы понять, как устроена нейросеть, представь себе простой поток: данные приходят на вход, проходят через цепочку преобразований – и на выходе получается результат. Все эти преобразования происходят внутри так называемых слоёв, а каждый слой состоит из нейронов.

Нейрон – это небольшая вычислительная ячейка. Он получает на вход числа, умножает их на свои «веса» (это такие коэффициенты важности), складывает результат, применяет к нему функцию – и передаёт дальше.

Это всё. На этом уровне – ничего магического. Только арифметика: сложение, умножение, и одна небольшая формула, которая называется функцией активации.

Зачем она нужна? Чтобы добавить нелинейность. Без неё сеть могла бы только проводить прямые линии – а нам нужно, чтобы она могла распознавать сложные формы, закономерности, переходы.

Теперь представь, что таких нейронов – десятки, сотни, тысячи. Они объединены в слои. Первый слой получает «сырые» данные (например, пиксели изображения или параметры клиента из таблицы). Последний слой – выдаёт результат (например, «да, это мошенническая транзакция» или «нет, это просто ошибка»). А всё, что между ними – внутренние уровни обработки, где происходит извлечение признаков, фильтрация, настройка.

Каждое соединение между нейронами имеет вес. Эти веса – и есть то, что «настраивает» модель в процессе обучения. Меняя веса, нейросеть учится. Если она делает ошибку – веса немного корректируются. Снова ошибка – снова корректируются. И так до тех пор, пока она не начнёт выдавать результат, близкий к правильному.

То есть нейросеть не «знает» правильный ответ заранее. Она учится на примерах, повторяя: вход ? обработка ? выход ? сравнение с правильным ? коррекция.

Именно в этом состоит её сила: она не зашита правилами, а настраивается под данные.

Глава 2. Первый шаг: перцептрон

История: перцептрон и забвение

Прежде чем появились нейросети, мир вычислений был строг до сухости. Машины оперировали правилами, заданными человеком, и не делали ни шага в сторону. Программист формулировал алгоритм, прописывал все возможные условия – и машина честно исполняла каждое из них. Такие системы назывались экспертными: если у пациента болит в правом боку, температура выше 38, и кровь показывает превышение лейкоцитов – выдать диагноз «аппендицит». Всё было чётко, жёстко и негибко.

В 1957 году молодой психолог и исследователь Фрэнк Розенблатт представил концепцию, которая казалась почти крамольной: а что если машина сможет учиться? Не просто следовать заранее заданным правилам, а сама, на основе примеров, выявлять закономерности и принимать решения. Так появился перцептрон – первая вычислительная модель, вдохновлённая устройством нейронов в мозге.

Модель была удивительно проста – входы, веса, порог активации – но результат ошеломил публику. Газеты писали: «Компьютер, который мыслит!», военные вкладывались в разработку машины Mark I Perceptron, предполагалось, что она сможет распознавать изображения, переводить речь и даже – научиться видеть.

Однако уже через десятилетие, в 1969 году, исследователи Марвин Минский и Сеймур Пейперт выпустили книгу с ироничным названием Perceptrons, где строго, с математическим хладнокровием доказали: эта модель беспомощна перед задачей XOR – элементарной логической функцией, которую невозможно разделить одной прямой. То есть перцептрон не может отличить, когда «одно из двух» – а не «оба» и не «ни одно».

Это стало ударом. Финансирование сократилось, интерес угас. В области искусственного интеллекта наступила так называемая «зима нейросетей»: десятилетие разочарования, когда направление считалось тупиковым, а серьёзные учёные переключались на другие методы.

Возвращение произошло в 1980-х, когда появились многослойные нейронные сети и алгоритм обратного распространения ошибки. Именно он позволил обучать сложные конструкции, а не только примитивные модели вроде перцептрона. И тогда оказалось: недостаток был не в самой идее, а в её ограниченной реализации. Перцептрон был шагом – первым, простым, детским – но шагом в верном направлении.

Зачем знать всё это сегодня? Потому что технологии меняются быстро, а принципы – медленно. Мода на алгоритмы приходит и уходит, но устройство нейросети, её слабые и сильные стороны, история её успехов и провалов – это то, что даёт понимание, а не просто навык.

Как он работает: простая формула и визуализация

Перцептрон – это не волшебство и не биологический нейрон в миниатюре. Это простейшая вычислительная схема, которая делает одну вещь: берёт набор чисел на входе, взвешивает их по важности, суммирует – и решает, включиться или нет. Как лампочка с чувствительным датчиком: загорится или нет, зависит от силы сигнала.

Представим: у нас есть три входа. Каждый – это какое-то значение: температура, цвет, вес, пиксель, уровень шума – неважно. Главное, что каждый вход имеет свой «вес»: число, которое показывает, насколько этот вход важен. Один – важен сильно, другой – почти не влияет. Мы перемножаем каждый вход на его вес и складываем всё вместе. Это и есть основа вычислений:

z = x?·w? + x?·w? + x?·w? + … + b

Здесь x – входы, w – веса, а b – так называемый «порог» или смещение (bias). Без него система была бы слишком жёсткой. Он позволяет сдвинуть линию принятия решения туда, где она лучше работает.

После сложения включается функция активации – она определяет, «зажжётся» ли выход. В самом простом варианте это порог: если сумма больше нуля – включаем, если нет – выключаем. Можно использовать другие варианты: сигмоиду, ReLU, гиперболический тангенс – всё зависит от задачи. Но в перцептроне изначально всё было просто: есть порог, есть решение.

Зачем всё это? Чтобы принимать бинарные решения. Например: эта точка – голубая или красная? Человек перед камерой – улыбается или нет? Письмо – спам или обычное? Если можно провести прямую, которая разделяет два класса на графике – перцептрон справится. Он как линейный «нож» – режет пространство ровно, без изгибов.

Обучение происходит по принципу «проба и ошибка». Перцептрон делает попытку: подаёт вход, считает выход. Потом сравнивает результат с правильным – и, если ошибся, чуть-чуть подправляет веса. Снова пробует. И снова. Так шаг за шагом он находит те веса, при которых будет реже ошибаться. Это напоминает обучение ребёнка: не сразу, не идеально, но с каждым разом всё точнее.

Хорошая визуализация – это двумерная плоскость, на которой точки двух классов – например, синие и красные – разбросаны как попало. Перцептрон пытается провести прямую, которая разделит их. Сначала – криво, потом – лучше. Каждая итерация приближает его к цели. И если задача действительно линейно разделима – он найдёт решение.

Но вот в чём его ограничение: он может провести только прямую. А если решение лежит «изгибом» – как в задаче XOR, где красные точки по диагонали, а синие – по другой? Тут перцептрон бессилен. Он не умеет гнуться. Это и стало главной причиной его забвения – и главной мотивацией к созданию многослойных сетей.

Практика: свой перцептрон на Python

Чтобы разобраться, как работает перцептрон, проще всего не читать сто объяснений, а один раз написать его руками. Причём без всяких библиотек, фреймворков и «магии» – только чистый Python. Наша цель – не эффективность, а понимание. Увидеть, как шаг за шагом модель учится принимать решения.

Исходная задача:

У нас есть набор точек на плоскости. Каждая точка имеет два координатных признака – x? и x?, и относится к одному из двух классов: 0 или 1. Мы хотим, чтобы перцептрон научился разделять эти два класса, проводя между ними прямую.

Код (минимальный):

```
import random

# инициализируем веса и порог
w1, w2 = random.random(), random.random()
b = random.random()
```

```
# функция активации (пороговая)
def activate(z):
return 1 if z > 0 else 0

# обучающая выборка: точки и правильные ответы
data = [
([0, 0], 0),
([0, 1], 0),
([1, 0], 0),
([1, 1], 1), # в этом примере – простая логика "и"
]

# цикл обучения
for epoch in range(20):
total_error = 0
for (x1, x2), target in data:
z = x1*w1 + x2*w2 + b
pred = activate(z)
error = target – pred
# корректировка весов и порога
w1 += error * x1 * 0.1
w2 += error * x2 * 0.1
b += error * 0.1
total_error += abs(error)
print(f"Эпоха {epoch}: ошибка {total_error}")
```

После нескольких итераций модель начинает правильно классифицировать точки. Мы видим, как веса изменяются, а ошибка уменьшается. Это и есть обучение.

Зачем всё это писать вручную:

* Ты на практике видишь, как формируется решение.
* Понимаешь, откуда берутся веса и зачем нужна активация.
* Начинаешь читать чужой код не как заклинание, а как набор знакомых шагов.
* Получаешь не «волшебную коробку», а предсказуемый инструмент.

Варианты простейших задач для мини-проекта:

1. Чётные и нечётные числа.

Вход – одно число, выход – 0 (чётное) или 1 (нечётное). Можно взять только последний бит: x % 2.

2. Классификация по возрасту.

Вход – возраст человека, выход – 0, если меньше 30, и 1, если 30 и старше. Это линейно разделимая задача – как раз то, что нужно для перцептрона.

3. Фильтр спама (простейший).

Каждое слово – это число. Например, «деньги» = 1, «подарок» = 1, а нейтральные слова – 0. Если сумма выше порога, считаем письмо спамом.

Во всех этих примерах перцептрон работает как простой классификатор. Он не понимает смысла, не строит гипотезы, не «думает». Он просто учится отличать одно от другого – по тем примерам, которые мы ему дали.

Глава 3. Векторизация и матричное мышление

Пока у нас была всего одна «нервная клетка», всё было просто: пара чисел на входе, пара весов, простое сравнение с порогом. Но как только мы хотим сеть посложнее – с несколькими

нейронами, с несколькими слоями – вручную это уже не пишется. Тут начинается настоящее матричное мышление.

Все современные нейросети – это по сути умножение матриц. Да, внутри – веса, функции активации, слои. Но с точки зрения кода и вычислений, всё сводится к тому, чтобы быстро и точно перемножать таблицы чисел. А значит, нужно перейти от списков и циклов к векторной форме.

Это не просто про скорость. Это про способ думать. Пока ты думаешь поэлементно – ты пишешь, как компьютер 80-х. Когда начинаешь мыслить матрицами – ты начинаешь мыслить, как сеть.

В этой главе:

* мы поймём, почему веса – это матрицы,

* увидим, как работает обратная связь (градиентный спуск, но без формул),

* подключим библиотеку NumPy и ускорим всё в десятки раз,

* напишем свою первую настоящую мини-сеть: с двумя слоями, с обучением и предсказанием.

Вот блок «Почему "веса" – это матрица» – строго, наглядно, без математики и с сохранением логики:

Почему «веса» – это матрица

Пока у нас один нейрон, всё просто: у него есть несколько входов, и каждому входу соответствует свой вес. Эти веса можно записать в виде списка чисел. Например, если у нас два входа, веса будут такие: [0.5, -0.2]. Мы умножаем входы на веса, складываем – и получаем результат.

Но что происходит, когда мы добавляем второй нейрон? А третий? Каждый из них тоже получает все те же входы – и у каждого свой набор весов. То есть теперь у нас не просто список, а таблица: в каждой строке – веса одного нейрона.

Это и есть матрица весов.

Если входы – это вектор (одна строка чисел), а каждый нейрон – это набор весов для этих входов (строка матрицы), то результат – это тоже вектор: столько чисел, сколько у нас нейронов в слое.

Например:

* у нас 3 входа: рост, вес и возраст;

* и 2 нейрона в первом слое: один отвечает за «молодость», другой – за «риск болезни»;

* тогда у нас 2 строки по 3 веса:

[0.1, -0.2, 0.5] ? нейрон 1

[0.3, 0.4, -0.1] ? нейрон 2

Когда мы подаём входной вектор, например [180, 75, 30], мы просто умножаем его на эту матрицу весов – и сразу получаем два числа: выходы этих нейронов.

Поэтому нейросеть – это не сложный набор условий, а просто цепочка матричных преобразований: входы ? веса ? результат.

Когда ты понимаешь это – ты готов к следующему шагу: строить не один нейрон, а слои, целые сети. И писать код уже в виде векторных операций.

Вот блок «Простейшая обратная связь: градиентный спуск (без формул!)» – без формул, но с пониманием сути:

Простейшая обратная связь: градиентный спуск (без формул!)

Любая обучающаяся система должна не только делать выводы, но и учиться на ошибках. Нейросеть – не исключение. И вся магия здесь – в том, как она находит ошибку и меняет свои веса.

Представь, что сеть – это стрелок, который пытается попасть в цель. Каждый выстрел – это её предсказание. После выстрела она узнаёт, насколько промахнулась – и сдвигает прицел. Снова стреляет, снова смотрит на ошибку. И так – до тех пор, пока не начнёт попадать.

Эта настройка прицела и есть обратная связь. А алгоритм, по которому сеть «крутит винтики» весов – это градиентный спуск.

Объясним проще:

* сеть сделала предсказание,
* посмотрела, насколько оно отличается от правильного ответа,
* прикинула, в какую сторону надо изменить каждый вес,
* изменила немного – не резко, а по чуть-чуть,
* повторила.

Так шаг за шагом, итерация за итерацией, она приближается к хорошему результату.

Важно: нейросеть не думает. Она не понимает, что делает. Она просто минимизирует ошибку – так, как если бы шарик катился вниз по склону, и каждый раз выбирал путь, где уклон чуть-чуть круче. Это и есть «спуск» – шаги по направлению уменьшения ошибки.

Чем меньше ошибка – тем ближе сеть к цели. Чем больше данных – тем точнее настройки.

И, как мы скоро увидим, этот принцип работает и для одной клетки, и для сотен слоёв. Главное – уметь передать ошибку назад и пересчитать веса. Вперед – предсказание, назад – обучение.

NumPy в помощь: ускоряем расчёты

NumPy – это одна из самых популярных библиотек для работы с массивами и матрицами в Python. Почему это важно для нейросетей? Потому что операции с большими объёмами данных требуют быстрых и эффективных вычислений. Вместо того чтобы работать с обычными списками Python (которые довольно медленные для числовых данных), NumPy позволяет нам создавать многомерные массивы, которые обрабатываются значительно быстрее.

Зачем нам NumPy?

1. Производительность: NumPy использует C и Fortran для выполнения операций, что в несколько раз быстрее, чем стандартные Python-списки.

2. Удобство работы с многомерными данными: Нейросети часто оперируют с данными, представляющими собой матрицы и тензоры. В NumPy мы можем легко работать с такими структурами.

3. Математические операции: В NumPy есть встроенные функции для работы с линейной алгеброй, статистикой и случайными числами, что позволяет не изобретать собственные решения для таких задач.

Основы работы с NumPy

1. Массивы (ndarray):

Массивы в NumPy – это основная структура данных, которая позволяет работать с числовыми данными. Давайте создадим простой массив:

2. import numpy as np
3. arr = np.array([1, 2, 3, 4])
4. print(arr)
5. Операции с массивами:

Операции с массивами NumPy выполняются поэлементно, что позволяет легко и быстро применять математические операции:

6. arr = np.array([1, 2, 3, 4])
7. arr_squared = arr ** 2
8. print(arr_squared)
9. Матричные операции:

Для нейросетей важна работа с матрицами. В NumPy это реализовано через умножение и другие линейные операции:

```
matrix_1 = np.array([[1, 2], [3, 4]])
matrix_2 = np.array([[5, 6], [7, 8]])
result = np.dot(matrix_1, matrix_2) # Умножение матриц
print(result)
```

Генерация случайных данных:

Нейросети часто нуждаются в случайных данных для инициализации весов и обучения. NumPy имеет функции для генерации случайных чисел:

```
random_weights = np.random.randn(3, 3) # Генерация случайных чисел
print(random_weights)
```

Почему это важно?

1. Базовый алгоритм обучения нейросетей: Градиентный спуск – это основной способ, с помощью которого нейросеть "учится". Все нейросети, от самых простых до самых сложных, используют его для настройки своих весов. Знание того, как он работает, помогает понять, как сеть находит лучшие решения, а также почему она может делать ошибки и как их исправить.

2. Контроль за процессом обучения: Когда ты понимаешь, как работает алгоритм, ты можешь влиять на процесс обучения. Например, ты можешь настроить скорость обучения (learning rate), чтобы ускорить или замедлить процесс, или остановить обучение в нужный момент, чтобы избежать переобучения (overfitting). Без такого понимания ты бы просто использовал алгоритм, не понимая, как он работает, и что делать, если что-то пошло не так.

3. Оптимизация и улучшение результатов: Знание того, как работает градиентный спуск, дает тебе возможность оптимизировать модель. Например, ты можешь выбрать другой тип градиентного спуска (например, стохастический градиентный спуск – SGD) или использовать более сложные методы, такие как адаптивный градиентный спуск (Adam), чтобы ускорить обучение и улучшить результаты.

4. Гибкость и адаптация: Если ты поймешь, как работает процесс обучения, тебе будет легче адаптировать его под свои задачи. Например, если твоя модель не работает как нужно, ты можешь изменить архитектуру, выбор функций активации, или другие параметры обучения, чтобы улучшить точность.

5. Понимание "магии" за моделями: Нейросети часто кажутся чем-то волшебным, потому что они могут решать сложные задачи, такие как распознавание изображений или генерация текста. Но на самом деле, за этим стоят простые математические операции, такие как градиентный спуск, которые подчиняются строгим законам. Знание этих принципов позволяет тебе не только лучше понимать уже существующие модели, но и создавать свои собственные, которые решают именно твои задачи.

6. Практическая уверенность: Когда ты понимаешь, как работает алгоритм, ты будешь увереннее в использовании нейросетей и в решении проблем. Это дает тебе больше контроля над проектом и позволяет не просто "играть" с готовыми моделями, а строить и внедрять эффективные решения, которые решают конкретные задачи.

Конец ознакомительного фрагмента.