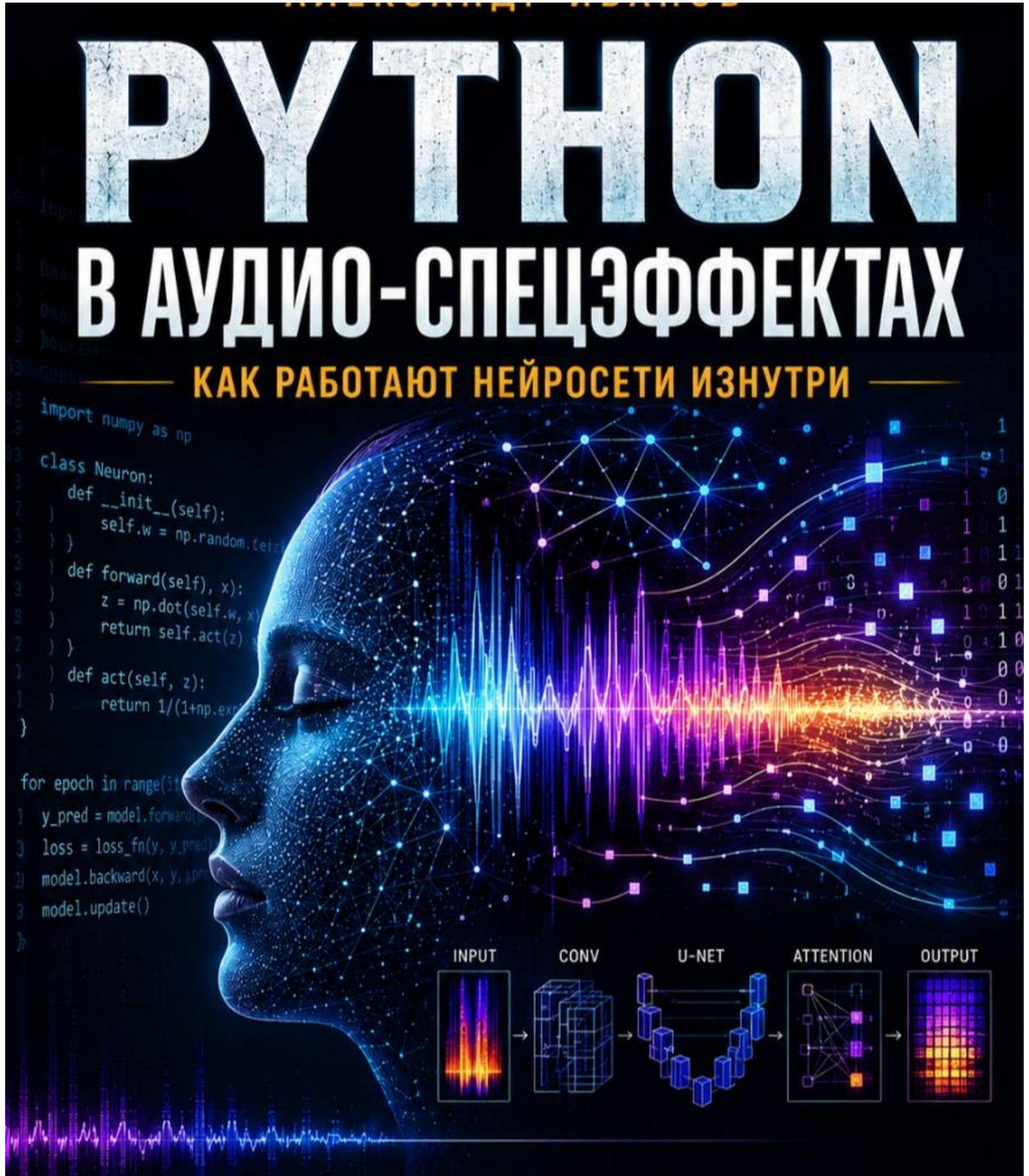
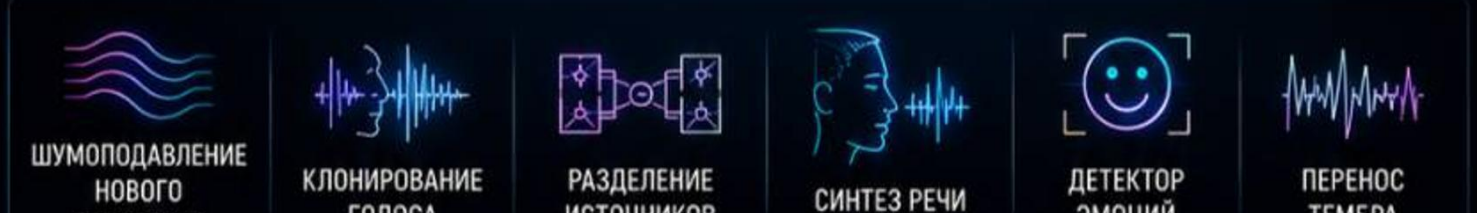
# PYTHON
# В АУДИО-СПЕЦЭФФЕКТАХ

## КАК РАБОТАЮТ НЕЙРОСЕТИ ИЗНУТРИ




ОТ ИСКУССТВЕННОГО НЕЙРОНА НА PYTHON
ДО СОЗДАНИЯ СОБСТВЕННЫХ АУДИОЭФФЕКТОВ
НА БАЗЕ ГЛУБОКОГО ОБУЧЕНИЯ



Александр Иванов

# Python в аудио-спецэффектах: как работают нейросети изнутри

«Автор»
2026

**Иванов А.**

Python в аудио-спецэффектах: как работают нейросети изнутри / А. Иванов — «Автор», 2026

Нейросети меняют мир обработки звука. Шумодавы, которые слышат лучше человека. Синтезаторы речи, неотличимые от живого голоса. Алгоритмы, разделяющие песню на дорожки за секунды. Всё это работает здесь и сейчас — и всё это можно понять и применить.Эта книга проведёт вас от простейшего нейрона до современных архитектур: свёрточных сетей, автокодировщиков, U-Net, attention, Tacotron. Вы не просто узнаете теорию — вы обучите модели для шумоподавления, разделения источников, синтеза речи, распознавания эмоций и создадите свой нейросетевой аудиоэффект.Никакой магии. Никакого чёрного ящика. Только код, интуиция и практика. Для тех, кто хочет не запускать готовые модели, а понимать их изнутри и создавать свои.

# Содержание

# Python в аудио-спецэффектах: как работают нейросети изнутри

## Предисловие

**Почему я написал эту книгу**

В первой книге мы учились обрабатывать звук. Мы загружали аудиофайлы, убирали шум, выравнивали громкость, меняли голос, микшировали с музыкой. Мы написали десятки скриптов, которые делали то, на что раньше уходили часы ручного труда. Это была книга-инструмент. Бери и делай.

Во второй книге мы пошли глубже. Мы разобрали математику, которая стоит за каждым эффектом. Мы написали преобразование Фурье с нуля. Мы спроектировали фильтры и свёртки. Мы поняли, как работает спектральное вычитание и адаптивная фильтрация. Мы создали свой синтезатор. Это была книга-понимание. Знай, что под капотом.

Теперь настало время для третьего шага. Для книги, которая смотрит в будущее. Потому что мир обработки звука меняется прямо сейчас. Традиционные алгоритмы — фильтры, свёртки, спектральное вычитание — постепенно уступают место нейросетям. Нейросетевое шумоподавление работает лучше классического. Нейросетевое разделение источников способно вытащить голос из шумной записи так, как не снилось никакому спектральному вычитанию. Синтез речи достиг уровня, когда сгенерированный голос неотличим от человеческого. И всё это происходит здесь и сейчас.

Но есть проблема. Большинство материалов по нейросетевой обработке звука написаны для исследователей. Они полны формул, графиков обучения, сравнений архитектур и ссылок на научные статьи. Они предполагают, что читатель уже знает и цифровую обработку сигналов, и машинное обучение. Обычный разработчик, звукорежиссёр или создатель контента, открывая такую статью, закрывает её через три минуты. Слишком сложно. Слишком много незнакомых терминов. Слишком большой разрыв между «я умею применять фильтры» и «я понимаю, как работает WaveNet».

Эта книга — мост через этот разрыв. Она написана для тех, кто прочитал первые две книги и хочет идти дальше. Или для тех, кто уже знает Python и основы обработки звука и хочет войти в мир нейросетей без боли и отчаяния.

**Для кого эта книга**

Эта книга для вас, если вы прочитали первые две книги серии — «Python для творческих: звук на твоей стороне» и «Цифровая обработка сигналов на Python: от инженера к разработчику». Вы умеете работать со звуком в коде. Вы понимаете, что такое спектрограмма, фильтр и свёртка. Теперь вы готовы к нейросетям.

Эта книга для вас, если вы не читали первые две книги, но уже программируете на Python и имеете базовое представление об обработке звука. Вы знаете, что такое частота дискретизации, амплитуда и спектр. Вы слышали про нейросети и, возможно, даже обучали пару моделей на картинках. Теперь вы хотите применить это знание к звуку.

Эта книга для вас, если вы звукорежиссёр, музыкант или создатель контента, который слышал про нейросетевые инструменты — iZotope RX, Adobe Podcast, ElevenLabs, — и хочет понимать, как они работают, чтобы использовать их осознанно. Вы не обязаны становиться исследователем. Но вы можете перестать верить в магию и начать понимать технологию.

Эта книга для вас, если вы разработчик, который хочет внедрить нейросетевую обработку звука в свой продукт. Вы хотите не просто скачать готовую модель, а понимать, какую

архитектуру выбрать, как подготовить данные и как интерпретировать результаты. После этой книги вы сможете обучать свои модели для своих задач.

**Как устроена эта книга**

Книга состоит из девяти глав. Мы начнём с самого простого — с одного искусственного нейрона. Мы напишем его на чистом Python, без библиотек, и научим его отличать громкий звук от тихого. Это даст нам интуицию о том, как машины учатся. Затем мы соберём из нейронов полносвязную сеть и научим её классифицировать звуки.

Дальше мы перейдём к главной идее нейросетевой обработки звука: спектрограмма как картинка. Мы узнаем, почему звук для нейросети — это не массив чисел, а изображение. Мы построим свёрточную нейросеть, которая видит на спектрограмме паттерны — форманты, обертоны, шум — и использует это для классификации и обработки.

Затем мы изучим автокодировщики — архитектуру, которая сжимает звук в компактное представление и восстанавливает обратно. Автокодировщики лежат в основе многих современных аудиоэффектов: они могут убрать шум, изменить тембр или выделить голос из смеси.

В пятой главе мы займёмся разделением источников — задачей, которая ещё десять лет назад казалась неразрешимой. Мы возьмём запись, где смешаны голос и музыка, и разделим их. Мы разберём архитектуры вроде Demucs и попробуем их в деле.

В шестой главе мы реализуем нейросетевое шумоподавление. Мы сравним его с классическим спектральным вычитанием из второй книги и увидим, насколько нейросети эффективнее.

В седьмой главе мы перейдём к синтезу речи. Мы узнаем, как работают голосовые ассистенты и системы клонирования голоса. Мы возьмём предобученную модель и заставим её говорить нашим голосом.

В восьмой главе мы займёмся детектором эмоций. Мы научим нейросеть определять по голосу, радуется человек, грустит, злится или спокоен. Это глава о том, как из аудиосигнала извлекается смысл.

В девятой главе мы создадим свой уникальный аудиоэффект с помощью нейросети. Это будет не готовая модель из интернета, а наша собственная архитектура, обученная под конкретную творческую задачу.

Бонусная глава соберёт всю трилогию воедино. Мы посмотрим на путь, который прошли: от первой книги, где мы просто вызывали готовые функции, до третьей, где мы создаём нейросетевые эффекты. Мы увидим, как три уровня — применение, понимание, создание — складываются в целостную картину.

Каждая глава построена по тому же принципу, что и во второй книге. Сначала — проблема и интуиция. Затем — теория шаг за шагом, без пропусков и без «очевидно, что». Затем — код. Мы будем писать нейросети с использованием PyTorch, но каждую строчку я объясню. Затем — визуализация: графики обучения, спектрограммы, результаты. Затем — «За кулисами»: история алгоритмов, связь с другими областями. Затем — «Лаборатория ошибок»: что ломается и как чинить. И наконец — «Творческое задание».

**Почему именно PyTorch**

Для нейросетей нам понадобится фреймворк глубокого обучения. Я выбрал PyTorch. Почему? Потому что он интуитивно понятен. Код на PyTorch читается как обычный Python. Вы можете поставить print() в любом месте и увидеть, что происходит внутри модели. Вы можете отлаживать нейросеть как обычную программу. Для обучения это неоценимо.

TensorFlow тоже мощный фреймворк, но его API более многословен, а отладка сложнее. Для наших целей — понять, как работают нейросети, а не просто запустить готовую модель, — PyTorch подходит идеально.

Если вы никогда не работали с PyTorch — не страшно. В первой главе мы установим его и напишем первый код. Всё, что вам нужно знать о тензорах, градиентах и обучении, я объясню по ходу дела.

**Что вам понадобится**

Вам понадобится Python 3.9 или новее и несколько библиотек: torch, torchaudio, librosa, numpy, soundfile, matplotlib. Всё это устанавливается через pip одной командой. В первой главе я дам точную инструкцию.

Вам понадобятся базовые знания Python и понимание основ обработки звука — то, что дают первые две книги. Если вы их не читали — ничего страшного, я буду напоминать ключевые концепты по мере необходимости. Но если вы встретите незнакомый термин вроде «спектрограмма» или «частота дискретизации» и захотите узнать о нём подробнее — загляните в приложения первой или второй книги.

Вам понадобится готовность к тому, что нейросети — это не всегда быстро. Обучение даже простой модели может занять несколько минут. Сложной — часы. Если у вас нет мощной видеокарты, не переживайте: все примеры в книге спроектированы так, чтобы работать на обычном ноутбуке. Там, где нужны серьёзные вычислительные ресурсы, я предложу предобученные модели.

И, как всегда, вам понадобится любопытство. Желание заглянуть под капот и понять, как устроена технология, которая меняет мир прямо сейчас.

# Пара слов перед стартом

Я пишу эту книгу в 2025 году. Нейросетевая обработка звука развивается с ошеломляющей скоростью. Каждый месяц появляются новые архитектуры, новые рекорды по качеству, новые применения. К тому моменту, когда вы будете читать эти строки, что-то уже может устареть.

Но есть вещи, которые не устаревают. Понимание принципов. Знание того, как нейросеть учится на данных. Умение выбрать архитектуру под задачу. Навык подготовки данных и интерпретации результатов. Этому и посвящена книга — не конкретным моделям, которые могут смениться через год, а фундаментальным идеям, которые останутся с вами надолго.

Первая книга дала вам инструменты. Вторая книга дала понимание. Третья книга даст вам способность создавать то, что ещё вчера казалось научной фантастикой.

Я помню момент, когда нейросети впервые меня по-настоящему удивили. Это было не в лаборатории и не на конференции. Я сидел дома, в наушниках, и запускал очередной эксперимент. У меня была запись голоса, сделанная на кухне. За окном шёл дождь, холодильник гудел, где-то на заднем плане играло радио. Классическое спектральное вычитание, которое я сам писал для второй книги, справлялось посредственно. Шум уходил, но голос становился глухим и неестественным, как из бочки. Я привык к этому результату. Я думал, что лучше сделать нельзя.

Потом я скачал предобученную нейросетевую модель шумоподавления — RNNoise, разработанную командой Mozilla. Она была крошечной, меньше мегабайта. Я подал на вход ту же самую запись. И услышал голос. Чистый, ясный, живой голос. Как будто человек сидел не на шумной кухне, а в студии с дорогим микрофоном. Никакой бочки. Никакой глухоты. Никаких артефактов. Только голос и тишина.

Я помню, как откинулся на спинку стула и несколько секунд просто смотрел в потолок. Я потратил недели на реализацию классических алгоритмов. Я понимал каждую строчку своего кода. Я знал, почему спектральное вычитание даёт музыкальный шум, почему гейт обрезает концы фраз, почему адаптивный фильтр не справляется с нестационарным шумом. И вот пришла крошечная нейросеть, которая ничего не знала о спектрах, фильтрах и теореме Найквиста — она просто смотрела на тысячи примеров шумной и чистой речи и училась делать из первого второе. И делала это лучше меня.

Это был переломный момент. Я понял, что нейросети — это не просто ещё один инструмент. Это другой уровень мышления о звуке. Классические алгоритмы опираются на наше понимание физики и математики звука. Мы проектируем фильтр, потому что знаем, как он влияет на частоты. Мы вычитаем спектр шума, потому что знаем, что шум стационарен. Нейросеть же не знает ничего. Она просто смотрит на данные и находит закономерности, которые мы, возможно, даже не можем сформулировать. И часто эти закономерности работают лучше наших теорий.

Но — и это важное «но» — слепая вера в нейросети так же опасна, как и их игнорирование. Нейросеть может научиться удалять шум, но может научиться и удалять полезный сигнал. Она может дать потрясающий результат на одних данных и полностью провалиться на других. Понимание того, что происходит внутри модели, критически важно. Нельзя просто взять готовую модель и надеяться на лучшее. Нужно знать, как она училась, на каких данных, какие у неё ограничения.

Именно этому и посвящена третья книга. Не просто запуску готовых моделей, а пониманию. Мы будем не только использовать нейросети, но и заглядывать внутрь: смотреть на функции потерь, анализировать кривые обучения, визуализировать активации. Мы будем задавать

вопросы: почему модель ошиблась? что она выучила? можно ли ей доверять? И мы будем писать свои модели — простые, понятные, но работающие.

Это третья ступень. Первая книга сделала вас практиком. Вторая — инженером. Третья сделает вас исследователем. Тем, кто не боится технологий будущего, потому что понимает их изнутри.

Поехали.

# Введение. Почему нейросети — это не магия

**О чём эта книга**

Откройте любой современный аудиоредактор. Найдите в нём функцию «шумоподавление». Скорее всего, вы увидите два варианта: классический и нейросетевой. Классический — это ползунки, пороги, параметры, которые нужно настраивать. Нейросетевой — одна кнопка. Нажали — шум исчез. Нажали ещё раз — и ваш голос звучит так, будто вы записывались в профессиональной студии, а не на кухне с воющим холодильником.

Теперь откройте любой сервис синтеза речи. Напечатайте текст. Выберите голос. Нажмите «озвучить». Через несколько секунд вы получите аудиофайл, в котором человеческий голос произносит ваш текст — с интонациями, паузами, естественными придыханиями. Вы можете дать послушать другу, и он не догадается, что это синтез.

Откройте приложение для видеозвонков. Оно убирает эхо, шум стройки за окном, лай собаки и плач ребёнка — всё в реальном времени, без задержки. Ваш собеседник слышит только ваш голос, чистый и ясный.

Всё это работает благодаря нейросетям. И всё это кажется магией. Вы нажимаете кнопку — получаете результат. Что происходит внутри? Непонятно. Почему результат такой хороший? Тоже непонятно. Почему иногда он всё-таки плохой? Тем более непонятно.

Эта книга написана для того, чтобы магия перестала быть магией. Чтобы нейросетевая обработка звука стала для вас не чёрным ящиком, а понятным инструментом. Таким же понятным, каким стали фильтры и свёртки после второй книги. Таким же привычным, каким стала нормализация громкости после первой.

Мы не будем погружаться в математические глубины машинного обучения на уровне исследовательских статей. Но мы не будем и просто запускать готовые модели, не понимая их устройства. Мы пойдём по среднему пути — тому самому, который сработал в первых двух книгах. Мы будем понимать достаточно, чтобы осмысленно применять нейросети, диагностировать проблемы и адаптировать модели под свои задачи. И мы будем писать код, который делает реальную работу со звуком.

**Три уровня работы со звуком**

В первой книге мы были практиками. У нас была задача: убрать шум, выровнять громкость, смонтировать подкаст. Мы брали готовые функции — librosa.effects.split, nr.reduce_noise, pyln.normalize.loudness — и получали результат. Мы не знали, что внутри, но мы знали, как применить. Это первый уровень: использование.

Во второй книге мы стали инженерами. Мы заглянули под капот. Мы узнали, что шумоподавление — это спектральное вычитание, что изменение голоса — это манипуляции со спектрограммой, что компрессор — это отслеживание огибающей и умножение на коэффициент усиления. Мы написали свои версии этих алгоритмов. Мы научились проектировать фильтры и синтезировать звук. Это второй уровень: понимание.

В третьей книге мы станем создателями нового. Мы узнаем, как работают нейросетевые алгоритмы, которые превосходят классические. Мы научимся обучать модели, которые убирают шум лучше спектрального вычитания, разделяют источники лучше адаптивных фильтров и синтезируют речь, неотличимую от человеческой. Мы не будем просто вызывать готовые модели — мы будем понимать их архитектуру, готовить для них данные, обучать и оценивать качество. Это третий уровень: создание интеллектуальных систем.

Эти три уровня не заменяют, а дополняют друг друга. Практик знает, какую кнопку нажать. Инженер знает, что за этой кнопкой происходит. Создатель знает, как сделать кнопку, которой ещё нет.

**Что такое нейросеть на самом деле**

Прежде чем мы начнём, давайте разберёмся с главным. Нейросеть — это не мозг. Это не искусственный интеллект в научно-фантастическом смысле. Это не существо, которое думает и осознаёт себя. Нейросеть — это математическая функция. Очень большая, очень сложная, с миллионами параметров, но — функция. Она принимает на вход числа и выдаёт на выходе числа.

В чём же отличие нейросети от обычной функции, скажем, от фильтра? Фильтр мы проектируем сами. Мы решаем: вот здесь будет частота среза, вот здесь — крутизна спада, вот такие коэффициенты. Мы понимаем, почему фильтр работает именно так. Нейросеть же никто не проектирует вручную. Её параметры — те самые миллионы чисел — находятся автоматически, в процессе обучения. Мы показываем нейросети тысячи примеров входных и выходных данных, и она постепенно подстраивает свои параметры так, чтобы для каждого входа выдавать правильный выход.

В этом и сила, и слабость нейросетей. Сила в том, что они могут научиться решать задачи, для которых у нас нет хороших алгоритмов. Мы не знаем, как точно описать правило удаления шума из произвольной записи — слишком много разных шумов и разных голосов. Но мы можем собрать тысячи примеров зашумлённых и чистых записей и показать их нейросети. И она найдёт закономерности, которые мы, возможно, даже не можем сформулировать словами.

Слабость в том, что мы не всегда понимаем, чему именно научилась нейросеть. Она может найти закономерность, которая работает на обучающих данных, но не работает в реальном мире. Она может научиться удалять не только шум, но и тихие согласные, которые похожи на шум. Она может «переобучиться» — запомнить обучающие примеры наизусть вместо того, чтобы вывести общее правило. Поэтому просто взять готовую модель недостаточно — нужно понимать, как она устроена и на каких данных обучена.

**Почему нейросети победили в обработке звука**

Классические алгоритмы обработки звука основаны на математических моделях. Мы предполагаем, что шум стационарен — и применяем спектральное вычитание. Мы предполагаем, что реверберация линейна — и применяем свёртку. Мы предполагаем, что голосовой сигнал периодичен на коротких интервалах — и применяем автокорреляцию для определения высоты тона.

Эти предположения работают. Но они работают не всегда. Реальный мир сложнее наших моделей. Шум не всегда стационарен — проезжающая машина или хлопок дверью меняют его характер за миллисекунды. Реверберация не всегда линейна — в маленькой комнате с мебелью отражения ведут себя сложнее, чем в пустом соборе. Голос не всегда периодичен — шёпот и придыхания не имеют основного тона вообще.

Нейросети не делают предположений. Они не знают, что такое стационарность, линейность или периодичность. Они просто смотрят на данные. Если в данных есть закономерность, нейросеть её выучит. Неважно, можем ли мы описать эту закономерность формулой. Неважно, понимаем ли мы её физический смысл. Если тысячи примеров показывают, что после определённого паттерна на спектрограмме идёт другой паттерн, нейросеть это запомнит и воспроизведёт.

Это не означает, что классические алгоритмы умерли. Они по-прежнему полезны, когда данных мало, когда нужна предсказуемость и интерпретируемость, когда важна скорость и малые вычислительные ресурсы. Но для задач, где качество важнее всего, нейросети сегодня — бесспорные лидеры.

**Как устроена эта книга**

Книга состоит из девяти глав. Мы начнём с самого простого — одного искусственного нейрона, — и дойдём до создания собственных нейросетевых аудиоэффектов.

В первой главе мы познакомимся с фундаментом: что такое искусственный нейрон, как из нейронов собирается сеть, что такое обучение и функция потерь. Мы напишем простую пол-

носвязную сеть на PyTorch и научим её классифицировать звуки — отличать речь от музыки, музыку от шума. Это даст нам базовый инструментарий, с которым мы будем работать всю книгу.

Во второй главе мы перейдём к ключевой идее нейросетевой обработки звука: спектрограмма как изображение. Мы узнаем, почему звук для нейросети удобно представлять в виде картинки, и как свёрточные нейросети, изначально придуманные для анализа изображений, оказались невероятно эффективны для анализа звука.

В третьей главе мы углубимся в свёрточные сети. Мы разберём, как они выделяют паттерны на спектрограмме — форманты гласных, обертоны, шумовые всплески, — и используем эти паттерны для классификации и обработки. Мы напишем свёрточную сеть, которая распознаёт музыкальные инструменты по звуку.

В четвёртой главе мы изучим автокодировщики — архитектуру, которая сжимает данные в компактное представление и восстанавливает обратно. Мы применим автокодировщик к спектрограммам и увидим, как он учится выделять самое важное и отбрасывать шум.

В пятой главе мы перейдём к разделению источников. Мы разберём архитектуры вроде Open-Unmix и Demucs, которые способны взять смесь голоса и музыки и разделить её на отдельные дорожки. Мы запустим предобученную модель и разберёмся, как она устроена внутри.

В шестой главе мы реализуем нейросетевое шумоподавление. Мы обучим свою модель, которая берёт зашумлённую запись и выдаёт чистую. Мы сравним результат с классическим спектральным вычитанием из второй книги и увидим разницу.

В седьмой главе мы займёмся синтезом речи. Мы узнаем, как работают модели Text-to-Speech, и научимся клонировать голос — создавать синтезированную речь, которая звучит как конкретный человек.

В восьмой главе мы переключимся на анализ эмоций в голосе. Мы обучим модель, которая по аудиозаписи определяет, радуется человек, грустит, злится или говорит нейтрально. Мы разберёмся с тем, какие признаки в голосе выдают эмоции.

В девятой главе мы создадим свой уникальный нейросетевой аудиоэффект. Мы придумаем творческую задачу — например, превращение голоса в звук музыкального инструмента или создание гибридного тембра, — и решим её с помощью нейросети.

Бонусная глава соберёт всю трилогию воедино. Мы посмотрим на путь, который мы прошли: от вызова готовых функций в первой книге до создания нейросетевых эффектов в третьей.

**Что вам понадобится**

Для работы с книгой вам нужен Python 3.9 или новее. Я предполагаю, что у вас уже установлены библиотеки из первых двух книг: numpy, librosa, soundfile, scipy, matplotlib. Если нет — установите их одной командой:

bash

```
pip install numpy librosa soundfile scipy matplotlib
```

Главная новая библиотека — PyTorch. Установка зависит от вашей системы, но для большинства пользователей подойдёт:

bash

```
pip install torch torchaudio
```

Если у вас есть видеокарта NVIDIA и вы хотите использовать её для ускорения обучения, установите версию с поддержкой CUDA. Инструкции есть на официальном сайте pytorch.org. Если видеокарты нет — не страшно. Все примеры в книге работают на обычном процессоре, просто обучение может занять чуть больше времени.

Вам также понадобится torchaudio — библиотека для работы со звуком, интегрированная с PyTorch. Она умеет загружать аудиофайлы, вычислять спектрограммы и применять аудио-эффекты — и всё это в виде тензоров PyTorch, готовых для подачи в нейросеть.

Для некоторых глав нам понадобятся предобученные модели. Я дам ссылки для скачивания и код для загрузки. Ничего покупать не нужно — всё бесплатно и открыто.

**Как читать эту книгу**

Главы построены последовательно: каждая следующая опирается на предыдущие. Если вы новичок в нейросетях, читайте по порядку. Если у вас уже есть опыт — можете переходить к интересующим главам, но имейте в виду, что в первой главе вводятся базовые понятия, которые используются дальше.

Каждая глава завершается творческим заданием. Не пропускайте их. Нейросети — это область, где понимание приходит через практику. Можно прочитать десять книг и всё равно не уметь обучить модель. А можно обучить одну модель своими руками — и понять больше, чем из всех книг вместе.

И, как всегда, экспериментируйте. Меняйте параметры. Ломайте код и чините. Нейросети — это не хрупкие конструкции. Они либо работают, либо нет. Когда не работают — это не провал, а приглашение к исследованию.

**Пара слов перед стартом**

Я помню свой первый опыт обучения нейросети. Это был простой перцептрон, который должен был отличать единицы от нулей на картинке восемь на восемь пикселей. Я запустил обучение и ждал. Процессор гудел, вентилятор ноутбука раскрутился до максимума. Через десять минут модель обучилась. Она правильно классифицировала аж семьдесят два процента примеров. Я был в восторге. Семьдесят два процента! Машина, которую я написал сам, научилась видеть!

Сейчас этот результат вызывает улыбку. Современные модели достигают точности, близкой к ста процентам, на гораздо более сложных задачах. Но то чувство — чувство, когда твоё творение учится, — остаётся неизменным. Я надеюсь, вы испытаете его в этой книге. Когда ваша первая нейросеть впервые правильно определит эмоцию в голосе или уберёт шум из записи — вы поймёте, о чём я говорю.

# Глава 1. От нейрона к сети: учим компьютер слышать

*«Нейросеть — не чёрный ящик. Это конструктор. Ты можешь собрать его сам»*

**О чём эта глава**

Всё, что мы будем делать в этой книге, строится на одном фундаменте: искусственных нейронных сетях. Шумоподавление, разделение источников, синтез речи, детектор эмоций — все эти впечатляющие технологии начинаются с одной простой идеи. С идеи о том, что компьютер может учиться на примерах. Не выполнять заранее написанные инструкции, а самостоятельно находить закономерности в данных.

В этой главе мы начнём с самого простого — с одного искусственного нейрона. Мы поймём, как он работает, как принимает решения и, главное, как он учится. Мы напишем нейрон на чистом Python, без единой строчки из библиотек глубокого обучения. Мы научим его различать громкий и тихий звук — задача простая, но она даст нам понимание механики обучения.

Затем мы соберём из нейронов полносвязную сеть. Мы узнаем, что такое скрытые слои, функции активации и обратное распространение ошибки. Мы познакомимся с PyTorch — фреймворком, который станет нашим основным инструментом на всю книгу. Мы напишем первую настоящую нейросеть, которая будет классифицировать аудиофрагменты — отличать речь от музыки, а музыку от шума.

К концу этой главы у вас будет работающая модель классификации звуков и, что важнее, понимание того, как она устроена внутри. Не магия. Математика и код.

**Зачем нейросети звук**

Прежде чем погружаться в нейроны и слои, давайте ответим на простой вопрос: зачем вообще применять нейросети к звуку? У нас уже есть классические алгоритмы. Фильтры. Свёртка. Спектральное вычитание. Они работают. Они понятны. Они не требуют обучающих данных и видеокарт. Зачем усложнять?

Представьте, что вы хотите написать программу, которая отличает лай собаки от плача ребёнка. С помощью классических алгоритмов вы бы пошли примерно таким путём. Вы бы записали десятки примеров лая и плача. Вы бы посмотрели на их спектрограммы. Вы бы заметили, что у лая есть характерные паттерны — короткие всплески в среднем диапазоне частот, у плача — более продолжительные звуки с модуляцией высоты. Вы бы придумали правила: если средняя частота выше X и длительность меньше Y, то это лай. Если есть периодическая модуляция с частотой Z, то это плач.

Этот подход работает. Но у него есть проблемы. Каждый новый звук требует нового набора правил. Плач одного ребёнка может сильно отличаться от плача другого. Лай таксы не похож на лай овчарки. Шум дождя можно спутать с шипением радио. Вам придётся придумывать и настраивать правила для каждого случая, и в какой-то момент сложность станет неуправляемой.

Нейросеть решает эту проблему иначе. Вы не придумываете правила. Вы собираете dataset — набор примеров: тысячу записей лая, тысячу записей плача, тысячу записей шума. Вы показываете их нейросети, и она сама находит закономерности, которые отличают один класс от другого. Ей не нужны формулы для средней частоты и длительности. Она сама выяснит, какие признаки важны, а какие нет.

Более того, нейросеть может найти закономерности, которые человек вообще не замечает. Мелкие детали в фазовом спектре. Тонкие корреляции между далёкими частотами. Паттерны, которые не видны на спектрограмме невооружённым глазом. Именно поэтому нейросетевое шумоподавление работает лучше классического, а нейросетевой синтез речи звучит естественнее, чем механическая сборка из фонем.

**Один нейрон: модель, которая учится**

Давайте начнём с самого простого строительного блока нейросети — искусственного нейрона. Биологический нейрон получает сигналы от других нейронов через дендриты, обрабатывает их в теле клетки и, если суммарный сигнал достаточно силён, посылает импульс по аксону. Искусственный нейрон работает похоже, но проще.

Искусственный нейрон получает на вход несколько чисел. Он умножает каждое число на свой вес — параметр, который определяет важность этого входа. Затем он суммирует все взвешенные входы, добавляет смещение — ещё один параметр — и применяет к результату функцию активации. Функция активации решает, насколько сильным будет выход нейрона. Если взвешенная сумма большая и положительная — выход близок к единице. Если маленькая или отрицательная — к нулю.

Формула нейрона предельно проста:

y = f(w1·x1 + w2·x2 + ... + wn·xn + b)

где x1, x2 — входы, w1, w2 — веса, b — смещение, f — функция активации, y — выход.

Веса и смещение — это то, что нейрон настраивает в процессе обучения. В начале они случайные, и нейрон выдаёт бессмысленный результат. Но у нас есть обучающие примеры — входы и правильные ответы для них. Мы подаём вход, смотрим на выход нейрона, сравниваем с правильным ответом и вычисляем ошибку. Затем мы чуть-чуть меняем веса так, чтобы ошибка уменьшилась. Повторяем тысячи раз. Постепенно нейрон учится давать правильные ответы.

**Пишем нейрон на Python**

Давайте реализуем эту идею в коде. Мы создадим класс для одного нейрона и научим его решать простейшую задачу: определять, громкий звук или тихий.

```
python
import numpy as np
class SingleNeuron:
"""
Один искусственный нейрон.
Принимает один входной признак и выдаёт бинарное предсказание.
"""
def __init__(self):
# Инициализируем вес и смещение случайными малыми значениями
self.weight = np.random.randn() * 0.1
self.bias = np.random.randn() * 0.1

def sigmoid(self, x):
"""Сигмоидная функция активации: превращает любое число в значение от 0 до 1."""
return 1.0 / (1.0 + np.exp(-x))

def forward(self, x):
"""
Прямой проход: вычисляет выход нейрона по входу x.
x — одно число (например, средняя громкость аудиофрагмента).
"""
z = self.weight * x + self.bias
return self.sigmoid(z)

def train_step(self, x, y_true, learning_rate=0.1):
"""
Один шаг обучения.
x — входное значение
```

```
y_true — правильный ответ (0 или 1)
learning_rate — скорость обучения
"""
# Прямой проход
y_pred = self.forward(x)

# Вычисляем ошибку (разница между предсказанием и истиной)
error = y_pred - y_true

# Вычисляем градиенты (насколько нужно изменить параметры)
# Для сигмоиды: производная = y_pred * (1 - y_pred)
d_weight = error * y_pred * (1 - y_pred) * x
d_bias = error * y_pred * (1 - y_pred)

# Обновляем параметры
self.weight -= learning_rate * d_weight
self.bias -= learning_rate * d_bias

return error
# Создаём нейрон
neuron = SingleNeuron()
# Готовим обучающие данные
# Представим, что мы измерили среднюю громкость нескольких аудиофрагментов
# Значения около 0.8 — громкий звук (метка 1)
# Значения около 0.2 — тихий звук (метка 0)
train_data = [
(0.15, 0.0), (0.22, 0.0), (0.18, 0.0), (0.25, 0.0), (0.10, 0.0), # тихие
(0.75, 1.0), (0.82, 1.0), (0.70, 1.0), (0.88, 1.0), (0.79, 1.0), # громкие
]
print("Обучение одного нейрона:")
print("Эпоха | Вес | Смещение | Пример | Предсказание | Истина | Ошибка")
print("-" * 70)
# Обучаем нейрон
for epoch in range(200):
total_error = 0
for x, y_true in train_data:
y_pred = neuron.forward(x)
error = neuron.train_step(x, y_true, learning_rate=0.5)
total_error += abs(error)
if epoch % 50 == 0 or epoch < 10:
x_example, y_example = train_data[0]
y_pred_example = neuron.forward(x_example)
print(f"{epoch:5d} | {neuron.weight:+.3f} | {neuron.bias:+.3f} | "
f"{x_example:.2f} | {y_pred_example:.4f} | {y_example:.1f} | "
f"{abs(y_pred_example - y_example):.4f}")
# Проверяем обученный нейрон
print("\nРезультаты после обучения:")
test_data = [0.12, 0.30, 0.65, 0.90, 0.20, 0.85]
for x in test_data:
```

```
pred = neuron.forward(x)
label = "ГРОМКИЙ" if pred > 0.5 else "ТИХИЙ"
print(f" Громкость: {x:.2f} -> {pred:.4f} ({label})")
```

Запустите этот код. Вы увидите, как в начале обучения нейрон выдаёт случайные предсказания — примерно 0.5 для любого входа. Постепенно вес и смещение подстраиваются. К концу обучения нейрон уверенно отличает тихие звуки от громких: для значений около 0.2 предсказание близко к нулю, для значений около 0.8 — близко к единице.

Разберём ключевые моменты. Сигмоидная функция активации превращает любое число в значение между нулём и единицей. Большое положительное число на входе даёт значение, близкое к единице. Большое отрицательное — близкое к нулю. Это удобно для бинарной классификации: мы интерпретируем выход как вероятность того, что вход принадлежит классу «1».

Обучение происходит через корректировку веса и смещения. Мы вычисляем ошибку — разницу между предсказанием и правильным ответом. Затем вычисляем градиенты — насколько нужно изменить параметры, чтобы ошибка уменьшилась. И делаем маленький шаг в направлении, противоположном градиенту. Это называется градиентным спуском.

**От нейрона к сети**

Один нейрон может решать только очень простые задачи — разделять данные прямой линией. В реальном мире границы между классами гораздо сложнее. Граница между речью и музыкой, между эмоцией радости и эмоцией грусти, между чистым голосом и зашумлённым — всё это нелинейные, запутанные границы, которые один нейрон не может описать.

Решение — объединить нейроны в сеть. В полносвязной сети нейроны организованы в слои. Каждый нейрон первого слоя получает входные данные. Каждый нейрон второго слоя получает выходы всех нейронов первого слоя. Каждый нейрон третьего — выходы второго. И так далее. Между слоями применяются нелинейные функции активации, и именно эти нелинейности позволяют сети описывать сложные границы.

Обучение сети происходит так же, как обучение одного нейрона, но с одним усложнением: ошибку нужно распространить от выходного слоя к входному через все промежуточные слои. Это называется обратным распространением ошибки. Мы вычисляем, как ошибка на выходе зависит от каждого веса в сети, и обновляем все веса одновременно в направлении уменьшения ошибки.

**Классификация звуков: речь, музыка, шум**

Давайте применим полносвязную сеть к реальной задаче классификации звуков. Мы научим сеть отличать речь от музыки и шума по набору признаков, извлечённых из аудиофрагментов. Для этого мы воспользуемся PyTorch — фреймворком, который берёт на себя вычисление градиентов и обновление параметров.

```
python
import torch
import torch.nn as nn
import torch.optim as optim
import librosa
import numpy as np
def extract_features(y, sr, frame_size=2048, hop_length=512):
"""
Извлекает признаки из аудиосигнала для подачи в нейросеть.
Возвращает список векторов признаков для каждого фрейма.
"""
features = []

# Разбиваем на фреймы
```

```
num_frames = (len(y) - frame_size) // hop_length + 1

for i in range(num_frames):
start = i * hop_length
frame = y[start:start + frame_size]

# Признак 1: RMS энергия (средняя громкость)
rms = np.sqrt(np.mean(frame ** 2))

# Признак 2: Zero-crossing rate (частота пересечений нуля)
zcr = np.sum(np.abs(np.diff(np.sign(frame)))) / (2 * len(frame))

# Признак 3: Спектральный центроид (центр тяжести спектра)
spectrum = np.abs(np.fft.fft(frame * np.hamming(len(frame))))
freqs = np.fft.fftfreq(len(frame), 1 / sr)
positive_freqs = freqs[:len(freqs)//2]
positive_spectrum = spectrum[:len(spectrum)//2]
if np.sum(positive_spectrum) > 0:
centroid = np.sum(positive_freqs * positive_spectrum) / np.sum(positive_spectrum)
else:
centroid = 0

# Признак 4: Спектральный спад (частота, ниже которой 85% энергии)
cumsum = np.cumsum(positive_spectrum ** 2)
if cumsum[-1] > 0:
rolloff_idx = np.where(cumsum >= 0.85 * cumsum[-1])[0][0]
rolloff = positive_freqs[rolloff_idx]
else:
rolloff = 0

features.append([rms, zcr, centroid / 1000, rolloff / 1000])

return np.array(features, dtype=np.float32)
class SoundClassifier(nn.Module):
"""
Полносвязная сеть для классификации звуков.
4 входных признака -> 16 скрытых нейронов -> 8 скрытых нейронов -> 3 класса
"""
def __init__(self):
super().__init__()
self.fc1 = nn.Linear(4, 16) # входной слой -> скрытый слой 1
self.fc2 = nn.Linear(16, 8) # скрытый слой 1 -> скрытый слой 2
self.fc3 = nn.Linear(8, 3) # скрытый слой 2 -> выходной слой (3 класса)
self.relu = nn.ReLU() # функция активации ReLU
self.softmax = nn.Softmax(dim=1) # для получения вероятностей

def forward(self, x):
x = self.relu(self.fc1(x))
x = self.relu(self.fc2(x))
```

```
x = self.softmax(self.fc3(x))
return x
# Создаём модель, функцию потерь и оптимизатор
model = SoundClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
print("Модель создана:")
print(model)
print(f"\nКоличество параметров: {sum(p.numel() for p in model.parameters())}")
```

Наша сеть принимает четыре признака для каждого аудиофрейма: среднюю громкость, частоту пересечений нуля, спектральный центроид и спектральный спад. Частота пересечений нуля показывает, насколько сигнал колеблется — у шума она высокая, у музыки средняя, у речи низкая. Спектральный центроид показывает, где сосредоточена энергия спектра — у шипящих звуков он высокий, у басовых — низкий. Спектральный спад показывает, насколько спектр сконцентрирован — у тональных звуков он низкий, у шумовых — высокий.

Сеть состоит из трёх слоёв. Первый слой принимает четыре признака и выдаёт шестнадцать скрытых представлений. Второй сжимает их до восьми. Третий выдаёт три числа — вероятности для каждого класса: речь, музыка, шум. Функция активации ReLU оставляет положительные значения без изменений и обнуляет отрицательные — это простая нелинейность, которая хорошо работает на практике.

**Обучение классификатора**

Теперь нам нужны данные для обучения. В идеале — тысячи размеченных аудиофрагментов. Для демонстрации мы создадим синтетические данные: сгенерируем признаки, похожие на те, что бывают у реальной речи, музыки и шума.

```
python
def generate_synthetic_data(n_samples=500):
"""
Генерирует синтетические данные для демонстрации обучения.
В реальном проекте здесь была бы загрузка реальных аудиофайлов.
"""
np.random.seed(42)
data = []
labels = []

for _ in range(n_samples):
# Класс 0: Речь
data.append([
np.random.normal(0.3, 0.1), # rms: умеренная громкость
np.random.normal(0.15, 0.05), # zcr: низкая частота пересечений нуля
np.random.normal(0.8, 0.2), # centroid: низкий центроид
np.random.normal(0.6, 0.3), # rolloff: умеренный спад
])
labels.append(0)

# Класс 1: Музыка
data.append([
np.random.normal(0.4, 0.15), # rms: средняя громкость
np.random.normal(0.25, 0.08), # zcr: средняя
np.random.normal(1.5, 0.5), # centroid: средний
```

```
np.random.normal(2.0, 1.0), # rolloff: выше
])
labels.append(1)

# Класс 2: Шум
data.append([
np.random.normal(0.2, 0.1), # rms: низкая громкость
np.random.normal(0.45, 0.1), # zcr: высокая
np.random.normal(2.5, 1.0), # centroid: высокий
np.random.normal(3.0, 1.5), # rolloff: высокий
])
labels.append(2)

# Перемешиваем
data = np.array(data, dtype=np.float32)
labels = np.array(labels)
shuffle_idx = np.random.permutation(len(data))
return data[shuffle_idx], labels[shuffle_idx]
# Генерируем данные
X, y = generate_synthetic_data(500)
# Разделяем на обучающую и тестовую выборки
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
# Преобразуем в тензоры PyTorch
X_train_tensor = torch.tensor(X_train)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
X_test_tensor = torch.tensor(X_test)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)
print(f"Обучающих примеров: {len(X_train)}")
print(f"Тестовых примеров: {len(X_test)}")
# Обучаем модель
model = SoundClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
num_epochs = 200
for epoch in range(num_epochs):
# Прямой проход
outputs = model(X_train_tensor)
loss = criterion(outputs, y_train_tensor)

# Обратный проход и оптимизация
optimizer.zero_grad()
loss.backward()
optimizer.step()

if epoch % 40 == 0 or epoch == num_epochs - 1:
# Оцениваем точность на тестовой выборке
with torch.no_grad():
```

```
test_outputs = model(X_test_tensor)
_, predicted = torch.max(test_outputs, 1)
accuracy = (predicted == y_test_tensor).float().mean()

print(f"Эпоха {epoch:3d}: Потери = {loss.item():.4f}, Точность = {accuracy:.2%}")
print("\nОбучение завершено!")
```

Запустите этот код. Вы увидите, как потери уменьшаются от эпохи к эпохе, а точность на тестовой выборке растёт. К концу обучения модель должна достигнуть точности выше девяноста процентов — это означает, что она хорошо научилась отличать три класса звуков друг от друга.

Разберём, что происходит при обучении. criterion = nn.CrossEntropyLoss() — функция потерь, которая измеряет, насколько предсказания модели отличаются от правильных ответов. optimizer = optim.Adam(model.parameters(), lr=0.01) — оптимизатор Adam, который сам подбирает величину шага для каждого параметра. loss.backward() — магия PyTorch: автоматически вычисляет градиенты для всех параметров модели. optimizer.step() — обновляет параметры в направлении уменьшения ошибки.

**Применяем модель к реальному звуку**

Теперь давайте проверим нашу модель на реальном аудиофайле. Мы загрузим запись, извлечём признаки и посмотрим, какие классы модель предсказывает для разных фреймов.

```
python
def classify_audio_file(filepath, model, label_names):
"""Классифицирует аудиофайл по фреймам."""
y, sr = librosa.load(filepath, sr=22050, mono=True)
features = extract_features(y, sr)

# Преобразуем в тензор
features_tensor = torch.tensor(features)

# Предсказываем
with torch.no_grad():
outputs = model(features_tensor)
_, predicted = torch.max(outputs, 1)

# Считаем статистику
unique, counts = np.unique(predicted.numpy(), return_counts=True)
total = len(predicted)

print(f"\nАнализ файла: {filepath}")
print(f" Всего фреймов: {total}")
for cls, count in zip(unique, counts):
print(f" {label_names[cls]}: {count} фреймов ({count / total:.1%})")

return predicted.numpy()
# Классифицируем тестовый файл (если есть) или создадим синтетический
label_names = ['Речь', 'Музыка', 'Шум']
# Создадим тестовый сигнал: первые 2 секунды — тон (музыка), затем шум, затем речь из файла
sr = 22050
t = np.linspace(0, 4.0, int(sr * 4.0), endpoint=False)
```

```
test_signal = np.zeros_like(t)
# Музыка: синусоида с обертонами
test_signal[:int(1.5 * sr)] = (np.sin(2 * np.pi * 440 * t[:int(1.5 * sr)]) +
0.5 * np.sin(2 * np.pi * 880 * t[:int(1.5 * sr)]))
# Шум
test_signal[int(1.5 * sr):int(3.0 * sr)] = np.random.randn(int(1.5 * sr)) * 0.3
# Речь (если есть файл) или ещё один тон другой частоты
try:
speech, _ = librosa.load('voice_sample.wav', sr=sr, mono=True)
test_signal[int(3.0 * sr):] = speech[:len(test_signal) - int(3.0 * sr)]
except:
test_signal[int(3.0 * sr):] = np.sin(2 * np.pi * 220 * t[int(3.0 * sr):])
import soundfile as sf
sf.write('test_mixed.wav', test_signal, sr)
print("Создан test_mixed.wav — смесь музыки, шума и речи")
# Классифицируем
predictions = classify_audio_file('test_mixed.wav', model, label_names)
```

Запустите код и посмотрите на результат. Модель должна правильно определить: первые фреймы — музыка, средние — шум, последние — речь. Это базовая классификация, но принцип, который мы здесь использовали, лежит в основе всех нейросетевых систем обработки звука.

**За кулисами: история перцептрона**

Первый искусственный нейрон — перцептрон — был предложен Фрэнком Розенблаттом в 1958 году. Это было аналоговое устройство размером с комнату, которое использовало резисторы и потенциометры для настройки весов. Розенблатт верил, что перцептроны станут основой искусственного интеллекта.

В 1969 году Марвин Минский и Сеймур Пейперт опубликовали книгу, в которой математически доказали, что однослойный перцептрон не может решить даже простейшую логическую задачу XOR — исключающее ИЛИ. Это доказательство охладило интерес к нейросетям на десятилетие — период, известный как «зима искусственного интеллекта».

Возрождение началось в 1980-х, когда Джеффри Хинтон и другие исследователи показали, что многослойные сети с нелинейными функциями активации способны решать XOR и множество других задач. Ключевым стало изобретение алгоритма обратного распространения ошибки, который позволял эффективно обучать глубокие сети. Сегодня тот самый XOR, который убил перцептрон, решается сетью из трёх нейронов за доли секунды.

**Лаборатория ошибок**

Модель не обучается, потери не уменьшаются. Проверьте скорость обучения. Слишком большая — модель «перепрыгивает» минимум и расходится. Слишком маленькая — обучение идёт слишком медленно. Попробуйте значения 0.1, 0.01, 0.001. Для Adam хороший стартовый диапазон — 0.001–0.0001.

Модель отлично работает на обучающих данных, но плохо на тестовых. Это переобучение. Модель запомнила обучающие примеры вместо того, чтобы выучить общие закономерности. Решения: уменьшить количество нейронов, добавить регуляризацию, увеличить количество обучающих данных.

Точность на одном классе высокая, на других — низкая. Данные несбалансированы. Если речевых примеров в десять раз больше, чем шумовых, модель будет предсказывать речь чаще. Решение: сбалансировать выборку или использовать взвешенную функцию потерь.

При попытке обучить модель на CPU всё работает очень медленно. Для маленьких моделей и небольших данных процессора достаточно. Если данных много, используйте GPU: в PyTorch это делается переносом модели и данных на устройство командой .to('cuda').

**Творческое задание**

Запишите три типа звуков: речь, музыку и шум — по несколько минут каждого. Извлеките из них реальные признаки с помощью extract_features. Обучите модель на реальных данных и сравните точность с синтетическими данными. Насколько реальные данные сложнее?

Добавьте четвёртый класс — например, тишину или звук природы. Измените модель так, чтобы она классифицировала четыре класса. Понадобится ли изменить архитектуру или достаточно просто переобучить выходной слой?

Визуализируйте границы решений. Возьмите два признака (например, RMS и спектральный центроид), обучите модель только на них и постройте график: точки разных цветов для разных классов и цветной фон, показывающий предсказания модели. Вы увидите, где модель ошибается и почему.

В следующей главе мы перейдём от абстрактных признаков к главному инструменту нейросетевой обработки звука — спектрограмме. Мы узнаем, почему звук для нейросети — это картинка, и как свёрточные сети, придуманные для анализа изображений, стали основой современных аудиоалгоритмов.

# Глава 2. Спектрограмма как картинка: звук глазами нейросети

**О чём эта глава**

В предыдущей главе мы сделали первый шаг в нейросетевую обработку звука. Мы обучили полносвязную сеть классифицировать аудиофрагменты по нескольким числовым признакам: громкости, частоте пересечений нуля, спектральному центроиду. Это работало, но у такого подхода есть фундаментальное ограничение. Мы сами решали, какие признаки важны. Мы говорили сети: «Смотри на среднюю громкость, на центроид, на частоту пересечений нуля». А что, если мы упустили что-то важное? Что, если ключевое отличие речи от музыки скрыто не в этих четырёх числах, а в каком-то тонком паттерне, который мы не догадались измерить?

В идеальном мире мы бы не выбирали признаки вручную. Мы бы отдали сети весь сигнал целиком и сказали: «Смотри сама. Найди закономерности». Но у полносвязной сети есть проблема: она не умеет работать с длинными последовательностями. Аудиофрагмент длиной в одну секунду при частоте дискретизации 22 050 Гц — это 22 050 чисел. Полносвязная сеть с таким количеством входов будет огромной, медленной и склонной к переобучению.

Решение было найдено в области, далёкой от звука — в компьютерном зрении. Там исследователи столкнулись с похожей проблемой: изображение содержит миллионы пикселей, и подавать их все на вход полносвязной сети неэффективно. Они придумали свёрточные сети — архитектуру, которая смотрит на изображение через маленькие скользящие окна и выделяет локальные паттерны. И кто-то заметил: спектрограмма звука — это тоже изображение. По горизонтали — время, по вертикали — частота, а цвет или яркость — энергия. И на этом изображении есть текстуры, формы, линии — всё то, что свёрточные сети отлично умеют анализировать.

В этой главе мы перейдём от ручных признаков к спектрограммам. Мы узнаем, как превратить звук в изображение, которое нейросеть может «видеть». Мы построим свёрточную сеть, которая анализирует спектрограмму, и научим её классифицировать звуки не по четырём числам, а по всему богатству частотно-временной картины.

**Почему спектрограмма — это картинка**

Спектрограмма — это визуальное представление звука. По горизонтальной оси откладывается время, по вертикальной — частота, а цвет или яркость каждой точки показывает, сколько энергии было на этой частоте в этот момент времени. Мы подробно разбирали спектрограммы во второй книге, в главе про STFT. Сейчас важно понять, почему это представление так хорошо подходит для нейросетей.

Когда человек смотрит на спектрограмму, он видит осмысленные структуры. Речь выглядит как серия горизонтальных полос — это основной тон и обертоны, — перемежающихся вертикальными тёмными полосами пауз. Гласные звуки выглядят как яркие области в определённых частотных диапазонах — формантах. Согласные — как короткие шумовые всплески. Музыка выглядит как множество параллельных горизонтальных линий на частотах, кратных основному тону. Шум — как равномерная текстура без выраженной структуры.

Свёрточная нейросеть, обученная на изображениях, умеет выделять текстуры, края, формы. Она находит маленькие паттерны — например, горизонтальную линию определённой толщины, — затем собирает их в более крупные структуры — например, набор параллельных линий, характерный для гласного звука, — и в итоге формирует высокоуровневое представление: «это речь». Всё это происходит автоматически, без нашего указания. Сеть сама решает, какие визуальные признаки важны для классификации.

**От сигнала к спектрограмме**

Давайте напишем код, который превращает аудиосигнал в спектрограмму — тензор, готовый для подачи в нейросеть.

```
python
import torch
import torchaudio
import librosa
import numpy as np
def audio_to_mel_spectrogram(y, sr, n_mels=128, n_fft=2048, hop_length=512):
"""
Превращает аудиосигнал в мел-спектрограмму.

Мел-шкала имитирует восприятие частоты человеческим ухом:
низкие частоты представлены подробнее, высокие — грубее.
"""
# Преобразуем в тензор PyTorch, если пришёл numpy-массив
if isinstance(y, np.ndarray):
y = torch.tensor(y, dtype=torch.float32)

# Создаём преобразователь
mel_transform = torchaudio.transforms.MelSpectrogram(
```

# Конец ознакомительного фрагмента.