ПрактикаГлава 4. Анализ данных и машинное обучение

§ 1. Тема 28А. Практическая работа с искусственной нейронной сетью

Ключевые слова:

- MNIST
- кросс-энтропия
- датасет
- пиксель
- градиентный спуск
- точность

Цель практической работы

В этой практической работе мы разберём задачу распознавания рукописных цифр от начала до конца. Работа выполняется в ноутбуке, который можно запустить в среде Google Colab. Для этого достаточно открыть ссылку и последовательно выполнить ячейки ноутбука. При запуске на собственном компьютере потребуется установить Python и импортировать все необходимые библиотеки. При запуске ноутбука программа автоматически загружает набор данных MNIST, содержащий изображения рукописных цифр и правильные ответы к ним.

Важно, что модель не просто «угадывает» цифру. Она получает изображение в виде набора чисел, обрабатывает его в слоях нейронной сети и выдаёт десять чисел. Каждое число соответствует одной цифре. Первое число показывает вероятность того, что на изображении представлена цифра 0, второе число соответствует цифре 1, и так далее до цифры 9. Ответом считается класс, которому соответствует наибольшая вероятность.

Как и любой эксперимент, практическая работа строится в три шага: гипотеза, проверка, вывод. Наша гипотеза такова: нейронная сеть, обученная на большом наборе примеров, будет правильно распознавать

и новые рукописные цифры, которые она не видела при обучении. Проверять гипотезу будем на тестовой выборке, то есть на примерах, не участвовавших в обучении. В конце сформулируем вывод: подтвердилась ли гипотеза и на каких примерах модель ошибается. Общий ход работы показан на рис. 29.1.

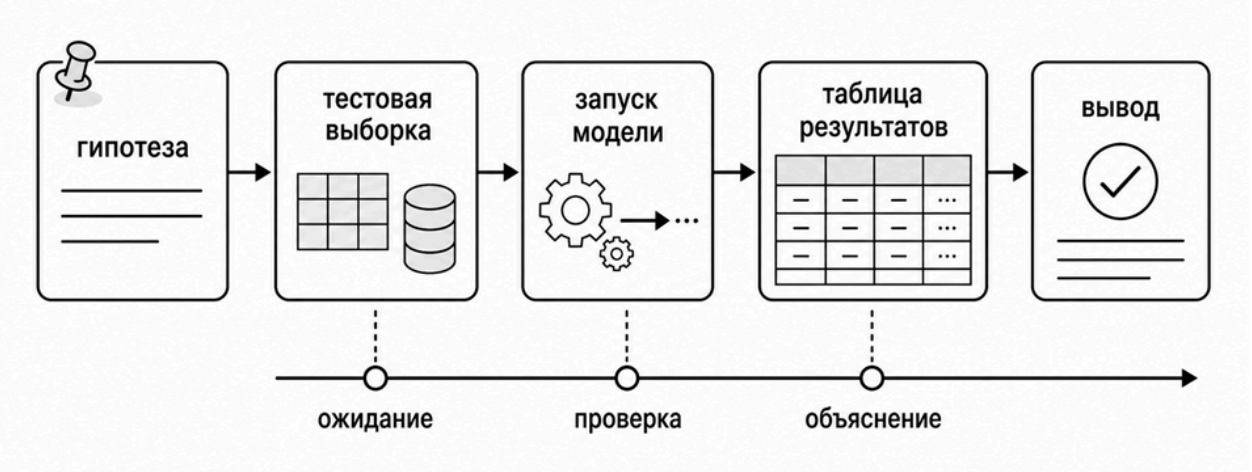

Рис. 29.1. Ход практической работы: от гипотезы к проверяемому выводу.

Главная цель работы состоит в том, чтобы понять полный ход эксперимента: как картинка становится вектором, какую функцию должна построить нейронная сеть, как устроены слои модели, какая функция ошибки используется при обучении и как проверить результат на тестовых данных.

Датасет MNIST

Для задачи распознавания цифр удобно использовать датасет MNIST. Это набор из 70 000 изображений рукописных цифр от 0 до 9. Каждое изображение имеет размер 28 × 28 пикселей и хранится в оттенках серого: значение каждого пикселя является целым числом от 0 до 255, и чем больше значение, тем светлее пиксель.

В ноутбуке датасет загружается одной командой:

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

Обратите внимание: данные сразу разделены на две части. Обучающая выборка x_train содержит 60 000 изображений, на которых модель будет подбирать параметры. Тестовая выборка x_test содержит 10 000 изображений. Эти изображения модель при обучении не видит, поэтому именно на них можно честно проверить готовую

модель. Перед обучением значения пикселей делят на 255, чтобы они попали в диапазон от 0 до 1:

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

При делении массива на число ошибка не возникает, потому что числовые массивы NumPy в Python поддерживают поэлементные арифметические операции. Поэтому число 255.0 делит каждое значение массива отдельно.

На рис. 29.2 показаны примеры изображений цифр из MNIST. Видно, что даже внутри одного класса цифры отличаются: разные люди пишут одну и ту же цифру по-разному. Поэтому модель должна не запоминать конкретные картинки, а находить закономерности, которые сохраняются у новых примеров.

Рис. 29.2. Примеры изображений из датасета MNIST: 10 классов цифр от 0 до 9.

Картинка как вектор

Человек видит изображение цифры как маленькую картинку. Модель машинного обучения работает с ним иначе: для неё это вектор признаков. Если развернуть матрицу 28 × 28 по строкам, получится вектор из 784 координат:

$$x = (x_1, x_2, \dots, x_{784}) \in \mathbb{R}^{784}$$

Слова «784-мерное пространство» звучат сложно, но смысл простой. Точка на прямой задаётся одним числом, точка на плоскости двумя числами, точка в обычном пространстве тремя числами. По той же логике изображение MNIST задаётся 784 числами. Представить такое

пространство глазами нельзя, но работать с ним можно как с обычным списком координат.

На рис. 29.3 одна и та же цифра показана в двух видах: как матрица интенсивностей и как вектор. Именно второе представление подаётся на вход многослойному персептрону.

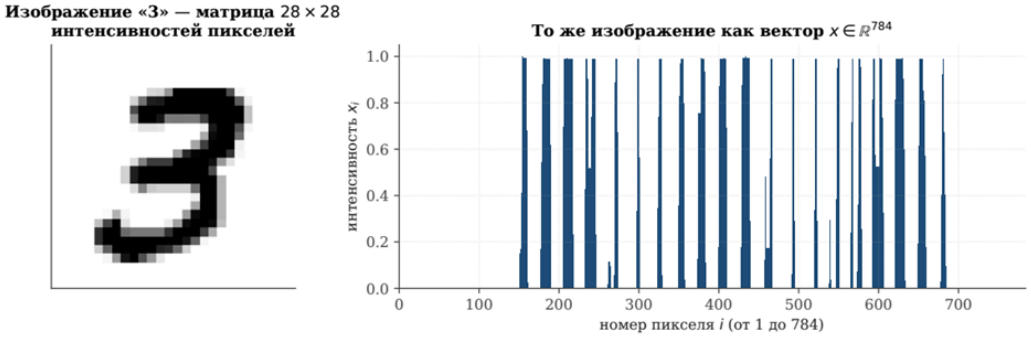

Рис. 29.3. Одно изображение цифры можно рассматривать как матрицу 28 × 28 и как вектор из 784 чисел.

Какую функцию строит модель

Нам нужно построить функцию, которая принимает на вход вектор из 784 чисел и возвращает вектор из 10 чисел:

$$f\colon \mathbb{R}^{784} \to \mathbb{R}^{10}.$$

На выходном слое модели вычисляются десять числовых значений, по одному для каждой цифры от 0 до 9. Функция *softmax* преобразует эти значения в десять чисел от 0 до 1, сумма которых равна 1. Поэтому их можно интерпретировать как оценки вероятности принадлежности изображения к соответствующим классам.

Функция *softmax* преобразует исходный вектор $\mathbf{x}=(x_1,x_2,\ldots,x_n)$ следующим образом:

$$\mathrm{softmax}(\mathbf{x}) = \left(\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_3}}{\sum_{i=1}^{n} e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}} \right)$$

где x_i — i-й элемент вектора $\mathbf{x}$. В задаче распознавания цифр $n=10$. Модель выбирает цифру, которой соответствует наибольшее значение в полученном векторе. Например, если наибольшее значение соответствует классу 3, модель считает, что на изображении, скорее всего, представлена цифра 3.

Для каждой картинки в обучающем наборе известна метка, то есть правильный ответ:

$$y \in \{0, 1, \dots, 9\}.$$

Обучающая выборка состоит из пар: картинка и её метка. Записывается это так:

$$\{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in \mathbb{R}^{784}, \quad y_i \in \{0, \dots, 9\}.$$

Здесь n обозначает число обучающих примеров; в нашем случае n = 60 000. Задача обучения состоит в том, чтобы по этим парам подобрать параметры модели. После обучения модель должна правильно работать не только на известных примерах, но и на новых изображениях, которые не входили в обучающий набор.

Архитектура многослойного персептрона

В ноутбуке используется простая нейронная сеть:

```
Input(shape=(28, 28))

Flatten()

Dense(128, activation='sigmoid')

Dense(10, activation='softmax')
```

В первой строке Input(shape=(28, 28)) задаётся размер входного изображения. Оно состоит из 28 × 28 пикселей. Слой Flatten() разворачивает эту матрицу в вектор из 784 чисел. Запись Dense обозначает полносвязный слой. В строке Dense(128, activation='sigmoid') создаётся слой из 128 нейронов с сигмоидальной функцией активации. Чем больше нейронов слой содержит, тем более сложные закономерности может обнаруживать модель, однако при этом обучение может занимать больше времени. В строке Dense(10, activation='softmax') создаётся выходной слой из 10 нейронов, по одному для каждой цифры от 0 до 9. Функция softmax преобразует значения на выходе этого слоя в оценки вероятности для десяти классов.

На рис. 29.4 показана эта архитектура: вход из 784 чисел, скрытый слой из 128 нейронов и выходной слой из 10 нейронов. Справа на рисунке приведён результат работы сети для одной картинки: наибольшую вероятность (73.2 %) получила цифра 3, заметные доли достались похожим на неё цифрам 8 и 9, а вероятности остальных классов близки к нулю.

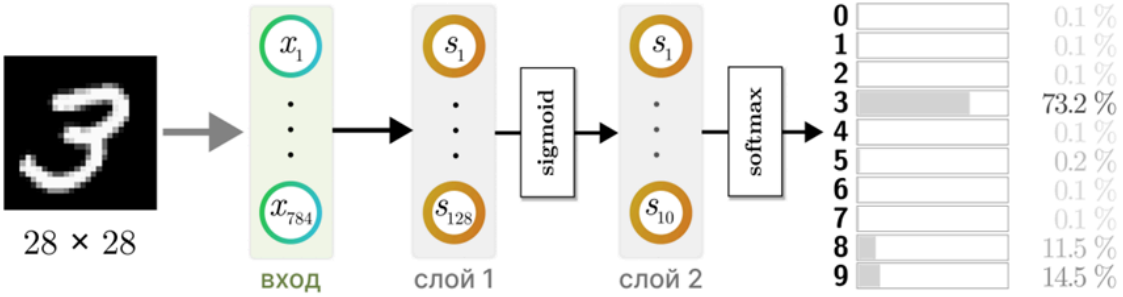

Рис. 29.4. Пример архитектуры нейронной сети для MNIST: вход из 784 чисел, скрытый слой из 128 нейронов, выходной слой из 10 нейронов.

Число параметров такой модели можно посчитать явно. Между входом и скрытым слоем есть $784 \cdot 128$ весов, между скрытым и выходным слоем есть $128 \cdot 10$ весов. Кроме того, у каждого из 128 нейронов скрытого слоя и 10 нейронов выходного слоя есть смещение. Поэтому:

$$784 \cdot 128 + 128 \cdot 10 + 128 + 10 = 101\,770.$$

Модель кажется маленькой, но в ней уже больше ста тысяч обучаемых параметров. В нашей модели их 101 770, поэтому подобрать параметры вручную невозможно: их подбирает численный метод.

Как параметры обучаются

Вспомним градиентный спуск (см. параграф 20), который мы уже обсуждали. В общем виде шаг обучения записывается так:

$$\theta_{k+1} = \theta_k - \alpha \nabla L(\theta_k).$$

Здесь α задаёт скорость обучения, а градиент $\nabla L(\theta_k)$ показывает направление наиболее быстрого роста ошибки. Поэтому алгоритм делает шаг в противоположную сторону, чтобы ошибка уменьшалась.

Чтобы выполнить шаг градиентного спуска в многослойной сети, недостаточно знать само значение ошибки на выходе. Нужно вычислить, как функция потерь изменяется при изменении каждого параметра сети, включая параметры первых слоёв. Иначе говоря, нужно найти градиент функции потерь по всем 101 770 параметрам. Эту задачу эффективно решает метод обратного распространения ошибки, или backpropagation: он последовательно вычисляет градиенты от выходного слоя к входному и показывает, как нужно изменить параметры сети, чтобы уменьшить функцию потерь.

В ноутбуке вся техническая часть скрыта внутри библиотеки Keras. Обучение запускается двумя командами:

```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model.fit(x_train, y_train, epochs=3, verbose=1)
```

Оптимизатор adam представляет собой усовершенствованный вариант градиентного спуска, который автоматически подстраивает величину шага. Функция потерь sparse_categorical_crossentropy соответствует кросс-энтропии для меток, записанных числами от 0 до 9. Метрика accuracy показывает долю правильных ответов. Параметр epochs=3 означает, что модель три раза пройдёт по всей обучающей выборке; один такой проход называется эпохой.

В запуске из ноутбука точность на обучающей выборке выросла примерно с 0.90 после первой эпохи до 0.96 после третьей. Модель действительно учится: ошибка уменьшается, а доля правильных ответов растёт.

Полезный эксперимент: ячейку обучения в ноутбуке можно сначала пропустить и сразу перейти к проверке. Необученная модель выдаёт почти равные вероятности, около 10 % на каждую цифру, то есть отвечает наугад. Затем запустите обучение и сравните распределения: станет наглядно видно, что именно даёт настройка параметров.

Проверка модели

После обучения модель проверяют на тестовых данных. В ноутбуке случайно выбирается одно изображение из x_test, и модель вычисляет вероятности:

```
pred = model.predict(img.reshape(1, 28, 28), verbose=0)

pred_label = np.argmax(pred)
```

Функция argmax возвращает номер класса с наибольшей вероятностью. Если максимум приходится на класс 7, модель отвечает: на изображении цифра 7.

В конкретном запуске из ноутбука получилась такая картина: вероятность 99.544 % у цифры 7, 0.358 % у цифры 9, 0.082 % у цифры 5, остальные ещё меньше. Нули в распечатке («0.000 %») появляются из-за округления: softmax никогда не даёт в точности нулевую вероятность. Этот пример показывает, почему полезно смотреть на распределение целиком: модель уверена в ответе 7, а небольшая доля у цифры 9 подсказывает, на какой класс изображение похоже больше всего.

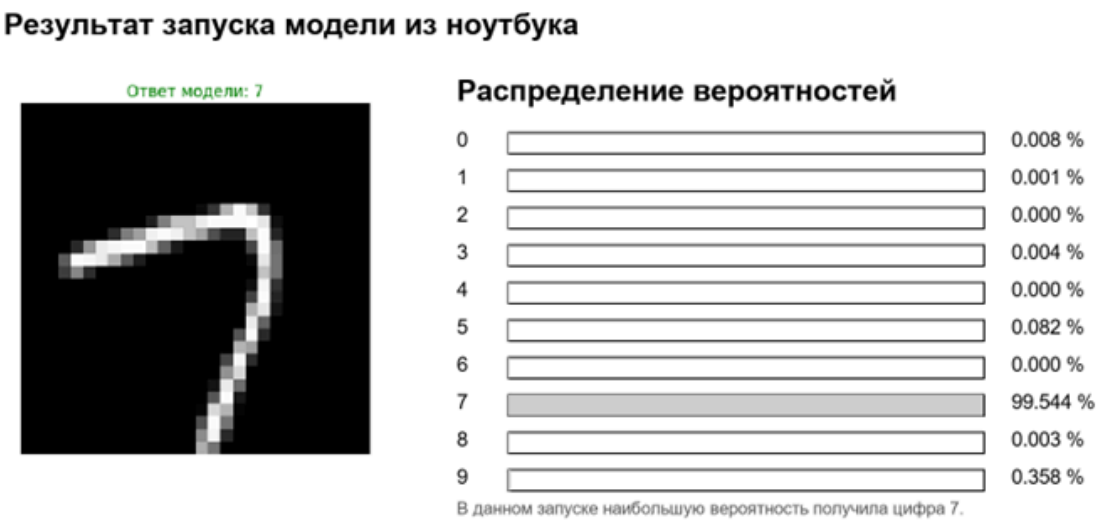

Рис. 29.5. Результат запуска модели из ноутбука: модель распознала цифру 7, справа показано распределение вероятностей по всем классам.

Один правильно распознанный пример не доказывает качество модели. Он лишь показывает, как устроен вывод. Для честной оценки модель нужно проверить на всей тестовой выборке.

Как оценить результат

Самой простой метрикой является accuracy, то есть доля правильных ответов. Если модель правильно распознала 950 изображений из 1000, accuracy равна 0.95. В Keras точность на всей тестовой выборке считается одной командой:

```
model.evaluate(x_test, y_test)
```

Но для отчёта одной цифры мало. Посчитайте точность отдельно по каждой цифре от 0 до 9: для этого достаточно сравнить ответы модели с метками y_test. Обычно некоторые цифры распознаются хуже других: например, цифра 3 часто путается с 8 и 9, цифра 1 с 7, а цифра 5 с 6. Выпишите несколько ошибок модели и постарайтесь объяснить, почему они возникли.

Полезно также проверить модель на изменённых изображениях: повернуть цифру или добавить помехи. Как показано на рис. 29.6, качество на простых примерах, на примерах с поворотом и на примерах с помехами может сильно различаться. Такой вывод честнее: модель хорошо работает на обычных изображениях, но хуже справляется с необычными.

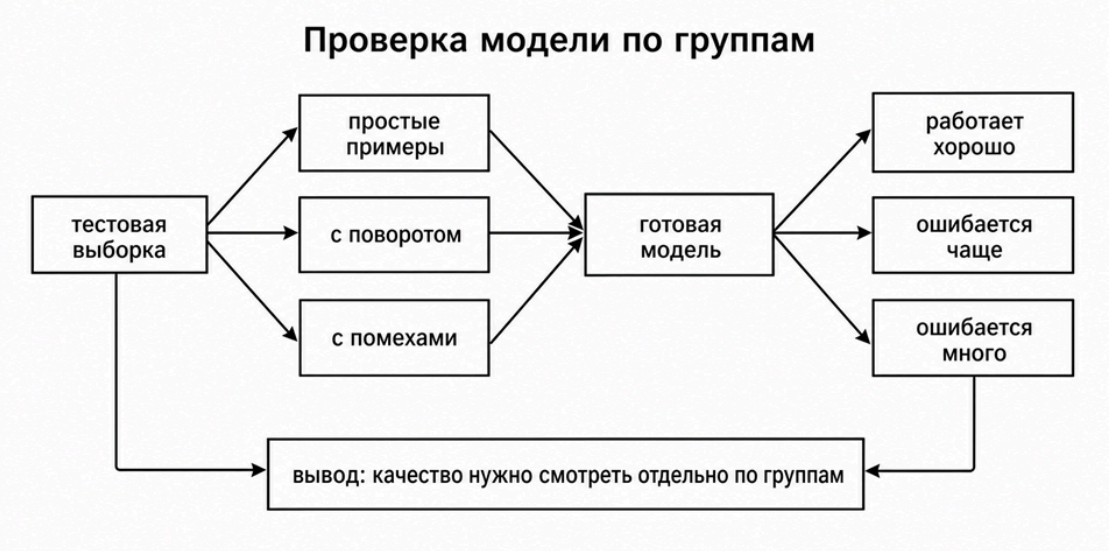

Рис. 29.6. Одну и ту же модель нужно проверять на разных группах примеров.

Отчёт по практической работе

Хороший отчёт содержит цель, описание датасета, архитектуру модели, функцию ошибки, способ обучения, пример вывода вероятностей и проверку качества. Важно не ограничиваться фразой

«нейросеть работает хорошо». Нужно указать, на каких данных это проверялось.

В отчёте можно придерживаться такой последовательности.

1. Описать датасет MNIST: 70 000 изображений, 10 классов, размер 28 × 28, разбиение на 60 000 обучающих и 10 000 тестовых примеров.
2. Объяснить, почему изображение превращается в вектор из 784 чисел.
3. Описать архитектуру: Flatten, 128 нейронов с sigmoid, 10 нейронов с softmax.
4. Записать функцию ошибки (кросс-энтропию).
5. Указать, что градиенты для обучения вычисляются методом обратного распространения ошибки.
6. Привести пример распределения вероятностей для одного изображения.
7. Посчитать accuracy на всей тестовой выборке с помощью model.evaluate(x_test, y_test) и разобрать типичные ошибки.

Достижения России: Михаил Бонгард и машины, которые учатся видеть

Михаил Моисеевич Бонгард (1924–1971) был советским учёным, одним из основоположников теории распознавания образов. Сегодня мы привыкли, что компьютер может найти лицо на фотографии, распознать рукописную цифру или определить объект на дороге. Но ещё в середине XX века сам вопрос звучал почти фантастически: можно ли научить машину «видеть» не как человека, а как вычислительную систему, которая выделяет признаки и на их основе принимает решение?

Бонгард занимался именно этой проблемой. Его интересовало не простое запоминание картинок, а способность находить в изображениях существенные признаки. В 1967 году вышла его книга «Проблема узнавания», позже переведённая на английский язык под названием Pattern Recognition. В ней рассматривалось, как машина может учиться различать объекты, выделять важную информацию и отбрасывать несущественную. По сути, речь шла о тех вопросах, которые остаются важными и для современных систем искусственного интеллекта.

Особую известность получили «задачи Бонгарда». В них даны две группы простых рисунков. Все рисунки в одной группе обладают некоторым общим свойством, а рисунки в другой группе этим свойством не обладают. Нужно понять, какое правило отличает одну группу от другой. Для человека многие такие задачи выглядят как интересные головоломки, но для машины они оказываются гораздо сложнее. Чтобы решить их, недостаточно просто сравнить пиксели. Нужно выделить форму, взаимное расположение элементов, симметрию, вложенность, количество объектов или другую скрытую закономерность.

Именно поэтому задачи Бонгарда до сих пор интересны исследователям искусственного интеллекта. Они показывают, что настоящее распознавание связано не только с вычислениями, но и с поиском признаков, обобщением и проверкой гипотез. Современные нейронные сети решают многие задачи компьютерного зрения гораздо успешнее ранних программ, но общий вопрос остаётся тем же: как перейти от изображения как набора чисел к пониманию его структуры?

Работы Бонгарда важны ещё и потому, что они показывают: исследования искусственного интеллекта активно развивались в нашей стране задолго до появления современных нейросетей и больших вычислительных мощностей. Советская школа распознавания образов внесла заметный вклад в развитие идей, без которых трудно представить сегодняшнее машинное обучение.

Вопросы и задания

1. Почему один удачный пример не доказывает качество модели?
2. Почему изображение 28×28 можно представить как последовательность?
3. Что содержат массивы x_train, y_train, x_test и y_test, и зачем данные разделяют на обучающую и тестовую выборки?
4. Зачем значения яркости пикселей делят на 255 перед обучением модели?
5. Зачем модели нужны скрытый слой из 128 нейронов и выходной слой из 10 нейронов? Какую роль выполняет функция softmax?
6. Как изменяются параметры модели в процессе обучения и зачем используются градиентный спуск и обратное распространение ошибки?