Экспертные системы и искусственный интеллект

**Лабораторная работа №1**

**Целью работы** является знакомство с тензорами как основным элементом для представления данных в нейронных сетях.

# 1. Представление данных для нейронных сетей

Все современные системы машинного обучения используют тензоры в качестве основной структуры данных. Тензоры являются фундаментальной структурой данных и представляют собой обобщение понятия «матрица» на произвольное количество измерений (в терминологии тензоров измерения часто называют осями).

## 1.1 Скаляры, векторы, матрицы и т.д.

Тензор, содержащий единственное число, называется скаляром (скалярным, или тензором нулевого ранга). В Numpy число типа float32 или float64 — это скалярный тензор (или скалярный массив). Определить количество осей тензора Numpy можно с помощью атрибута ndim; скалярный тензор имеет 0 осей (ndim == 0). Количество осей тензора также называют его рангом. Пример скаляра в Numpy:

>>> import numpy as np  
>>> x = np.array(12)  
>>> x  
array(12)  
>>> x.ndim  
0

Одномерный массив чисел называют вектором, или тензором первого ранга. Тензор первого ранга имеет единственную ось. Далее приводится пример вектора в Numpy:

>>> x = np.array([12, 3, 6, 14])  
>>> x  
array([12, 3, 6, 14])  
>>> x.ndim  
1

Этот вектор содержит пять элементов и поэтому называется пятимерным вектором. Не путайте пятимерные векторы с пятимерными тензорами! Пятимерный вектор имеет только одну ось и пять значений на этой оси, тогда как пятимерный тензор имеет пять осей (и может иметь любое количество значений на каждой из них).

Мерность может обозначать или количество элементов на данной оси (как в случае с пятимерным вектором), или количество осей в тензоре (как в пятимерном тензоре), что иногда может вызывать путаницу. В последнем случае технически более корректно говорить о тензоре пятого ранга (ранг тензора совпадает с количеством осей), но, как бы то ни было, для тензоров используется неоднозначное обозначение: пятимерный тензор.

Массив векторов — это матрица, или двумерный тензор. Матрица имеет две оси (часто их называют строками и столбцами). Матрицу можно представить как прямоугольную таблицу с числами. Вот пример матрицы в Numpy:

>>> x = np.array([[5, 78, 2, 34, 0],  
[6, 79, 3, 35, 1],  
[7, 80, 4, 36, 2]])  
>>> x.ndim  
2

Элементы на первой оси называют строками, а на второй — столбцами. В предыдущем примере [5, 78, 2, 34, 0] — это первая строка матрицы x, а [5, 6, 7] — ее первый столбец.

Если упаковать такие матрицы в новый массив, получится трехмерный тензор, который можно представить как числовой куб. Ниже приводится пример трехмерного тензора в Numpy:

>>> x = np.array([[[5, 78, 2, 34, 0],  
[6, 79, 3, 35, 1],  
[7, 80, 4, 36, 2]],  
[[5, 78, 2, 34, 0],  
[6, 79, 3, 35, 1],  
[7, 80, 4, 36, 2]],  
[[5, 78, 2, 34, 0],  
[6, 79, 3, 35, 1],  
[7, 80, 4, 36, 2]]])  
>>> x.ndim  
3

Упаковав трехмерные тензоры в массив, вы получите четырехмерный тензор и т. д. В глубоком обучении чаще всего используются тензоры от нулевого ранга до четырехмерных, но иногда, например при обработке видеоданных, дело может дойти и до пятимерных тензоров.

## 1.2 Ключевые атрибуты

Тензор определяется тремя ключевыми атрибутами:

* Количество осей (ранг) — например, трехмерный тензор имеет три оси, а матрица — две. В библиотеках для Python, таких как Numpy, этот атрибут тензоров имеет имя ndim.
* Форма — кортеж целых чисел, описывающих количество измерений на каждой оси тензора. Например, матрица в предыдущем примере имеет форму (3, 5), а трехмерный тензор имеет форму (3, 3, 5). Вектор имеет форму с единственным элементом, например (5,), тогда как скаляр имеет пустую форму ().
* Тип данных (обычно в библиотеках для Python ему дается имя dtype) — это тип данных, содержащихся в тензоре; например, тензор может иметь тип float32, uint8, float64 и др. В редких случаях можно встретить тензоры типа char. Обратите внимание, что в Numpy (и в большинстве других библиотек) отсутствуют строковые тензоры, потому что тензоры хранятся в заранее выделенных, непрерывных сегментах памяти и строки, будучи сущностями с изменяющейся длиной, препятствуют использованию такой реализации.

## 1.3 Манипулирование тензорами с помощью Numpy

Операция выбора конкретного элемента в тензоре называется получением среза тензора. Давайте посмотрим, какие операции получения среза тензора можно использовать с массивами Numpy.

Следующий пример извлекает цифры с 10-й до 100-й (100-я цифра не включается в срез) и помещает их в массив, имеющий форму (90, 28, 28):

>>> my\_slice = train\_images[10:100]

>>> print(my\_slice.shape)

(90, 28, 28)

Это эквивалентно более подробной форме записи, в которой определяются начальный и конечный индексы среза для каждой оси тензора. Обратите внимание, что : эквивалентно выбору всех элементов на оси:

>>> my\_slice = train\_images[10:100, :, :]

>>> my\_slice.shape

(90, 28, 28)

>>> my\_slice = train\_images[10:100, 0:28, 0:28]

>>> my\_slice.shape

(90, 28, 28)

В общем случае можно получить срез между любыми двумя индексами по каждой оси тензора. Например, вот как можно выбрать пикселы из области 14 × 14 в правом нижнем углу каждого изображения:

my\_slice = train\_images[:, 14:, 14:]

Допускается использовать и отрицательные индексы. Так же как отрицательные индексы в списках на Python, они будут откладываться от конца текущей оси.

1.4 Пакеты данных

В общем случае первая ось (ось с индексом 0, потому что нумерация начинается с 0) во всех тензорах, с которыми вам придется столкнуться в обучении, будет осью образцов (иногда ее называют измерением образцов).

Кроме того, модели глубокого обучения не обрабатывают весь набор данных целиком; они разбивают его на небольшие пакеты. Если говорить конкретнее, вот один пакет из примера с изображениями цифр MNIST, имеющий размер 128:

batch = train\_images[:128]

А вот следующий пакет:

batch = train\_images[128:256]

А вот n-й пакет:

batch = train\_images[128 \* n:128 \* (n + 1)]

При рассмотрении таких пакетных тензоров первую ось (ось с индексом 0) называют осью пакетов, или измерением пакетов.

## Примеры тензоров с данными

Чтобы было понятнее, перечислим несколько примеров тензоров с данными, которые могут встретиться вам в будущем. Данные, которыми вам придется манипулировать, почти всегда будут относиться к одной из следующих категорий:

* векторные данные — двумерные тензоры с формой (образцы, признаки);
* временные ряды или последовательности — трехмерные тензоры с формой (образцы, метки времени, признаки);
* изображения — четырехмерные тензоры с формой (образцы, высота, ширина, цвет) или с формой (образцы, цвет, высота, ширина);
* видео — пятимерные тензоры с формой (образцы, кадры, высота, ширина, цвет) или с формой (образцы, кадры, цвет, высота, ширина).

**Векторные данные** - наиболее часто встречающаяся форма данных. В таких наборах каждый образец может быть представлен вектором, а пакет, соответственно, двумерным тензором (то есть массивом векторов), где первая ось — это ось образцов, а вторая — ось признаков.

Рассмотрим два примера.

* Актуарный набор данных с информацией о людях, где для каждого человека указываются возраст, почтовый индекс и доход. Каждый человек характеризуется вектором с тремя значениями, соответственно, весь набор данных, описывающий 100 000 человек, можно сохранить в двумерном тензоре с формой (100000, 3).
* Коллекция текстовых документов, где каждый документ представлен количеством повторений каждого слова (из словаря с 20 000 наиболее употребительных слов). Каждый документ можно представить как вектор с 20 000 значений (по одному счетчику на каждое слово из словаря), соответственно, весь набор данных, описывающий 500 документов, можно сохранить в двумерном тензоре с формой (500, 20000).

Всякий раз, когда время (или понятие **последовательной упорядоченности**) играет важную роль в ваших данных, такие данные предпочтительнее сохранять в трехмерном тензоре с явной **осью времени**. Каждый образец может быть представлен как последовательность векторов (двумерных тензоров), а сам пакет данных — как трехмерный тензор (рисунок 1).

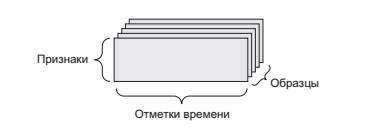
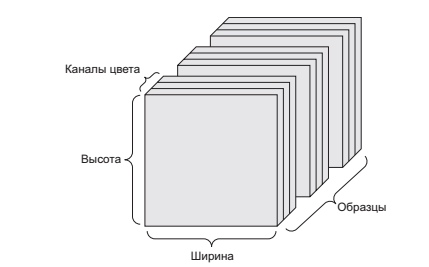


Рисунок 1

В соответствии с соглашениями, ось времени всегда является второй осью (осью с индексом 1). Рассмотрим несколько примеров.

* Набор данных с ценами акций. Каждую минуту мы сохраняем текущую цену акций, а также наибольшую и наименьшую цены за минувшую минуту. То есть каждая минута представлена трехмерным вектором, весь торговый день — двумерным тензором с формой (390, 3) (где 390 — длительность торгового дня в минутах), а данные за 250 дней — трехмерным тензором с формой (250, 390, 3). В данном случае каждый образец представляет данные за один торговый день.
* Набор данных с твитами, где каждый твит кодируется последовательностью из 280 символов из алфавита со 128 уникальными символами. В данном случае каждый символ можно закодировать как двоичный вектор со 128 элементами (содержит нули во всех элементах, кроме элемента с индексом, соответствующим номеру символа в алфавите, в который записывается 1). При такой организации каждый твит можно представить как двумерный тензор с формой (280, 128), а набор с миллионом твитов — как тензор с формой (1000000, 280, 128).

Обычно **изображения** имеют три измерения: высоту, ширину и цвет. Даже при том, что черно-белые изображения имеют только один канал цвета и могли бы храниться в двумерных тензорах, по соглашениям тензоры с изображениями всегда имеют три измерения, где для черно-белых изображений отводится только один канал цвета. Соответственно, пакет со 128 черно-белыми изображениями, имеющими размер 256 × 256, можно сохранить в тензоре с формой (128, 256, 256, 1), а пакет со 128 цветными изображениями — в тензоре с формой (128, 256, 256, 3).



В отношении форм тензоров с изображениями существует два соглашения: соглашение канал следует последним и соглашение канал следует первым.

**Видеоданные** — один из немногих типов данных, для хранения которых требуются пятимерные тензоры. Видео можно представить как последовательность кадров, где каждый кадр — цветное изображение. Каждый кадр можно сохранить в трехмерном тензоре (высота, ширина, цвет), соответственно, их последовательность можно сохранить в четырехмерном тензоре (кадры, высота, ширина, цвет), а пакет разных видеороликов — в пятимерном тензоре с формой (образцы, кадры, высота, ширина, цвет).

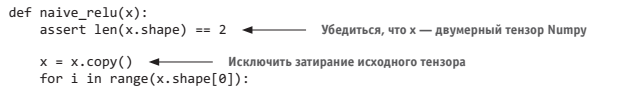
Например, 60-секундный видеоклип с разрешением 144 × 256 и частотой 4 кадра в секунду будет состоять из 240 кадров. Для сохранения пакета из четырех таких клипов потребуется тензор с формой (4, 240, 144, 256, 3). То есть 106 168 320 значений! Если предположить, что dtype тензора определен как float32, тогда для хранения каждого значения понадобится 32 бита, то есть для хранения всего тензора — 405 Мбайт.

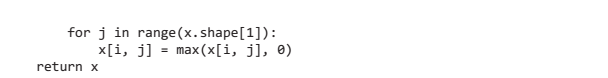
# 2 Операции с тензорами

Так как любую компьютерную программу можно свести к небольшому набору двоичных операций с входными данными (И, ИЛИ, НЕ и др.), все преобразования, выполняемые глубокими нейронными сетями при обучении, можно свести к горстке операций с тензорами, применяемых к тензорам с числовыми данными. Например, тензоры можно складывать, перемножать и т. д.

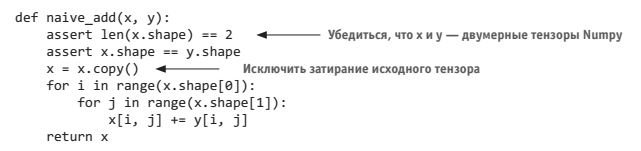
## 2.1 Поэлементные операции

Операция relu (эквивалентна max(x, 0)) и сложение — это поэлементные операции: операции, которые применяются к каждому элементу в тензоре по отдельности. То есть эти операции поддаются массовому распараллеливанию (векторизации, термин пришел из архитектуры векторного процессора суперкомпьютера периода 1970–1990-х). Для реализации поэлементных операций на Python можно использовать цикл for, как в следующем примере реализации операции relu:





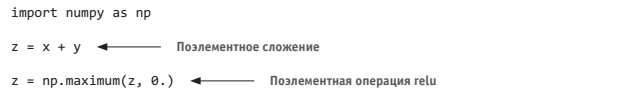
Точно так же реализуется сложение:



Следуя тому же принципу, можно реализовать поэлементное умножение, вычитание и т. д.

При работе с массивами Numpy можно пользоваться уже готовыми, оптимизированными реализациями этих операций, доступными в виде функций из пакета Numpy, которые сами делегируют основную работу реализациям базовых подпрограмм линейной алгебры (Basic Linear Algebra Subprograms, BLAS), если они установлены (конечно же, они должны быть у вас установлены). BLAS — это комплект низкоуровневых, параллельных и эффективных процедур для вычислений с тензорами, которые обычно реализуются на Fortran или C.

Иными словами, при использовании Numpy поэлементные операции можно записывать, как показано ниже, и они будут выполняться почти мгновенно:



## 2.2 Расширение

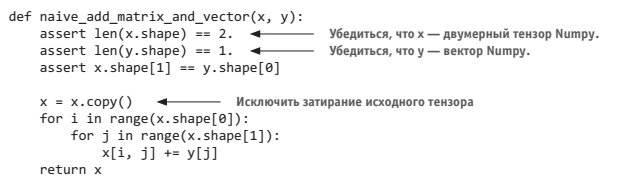
Наша предыдущая реализация naive\_add поддерживает только сложение двумерных тензоров с идентичными формами. Но в слое Dense, представленном выше, мы складывали двумерный тензор с вектором. Что происходит при сложении, когда формы складываемых тензоров различаются?

Когда это возможно и не вызывает неоднозначности, меньший тензор расширяется так, чтобы его новая форма соответствовала форме большего тензора. Расширение выполняется в два этапа:

1. В меньший тензор добавляются оси (называются осями расширения), чтобы значение его атрибута ndim соответствовало значению этого же атрибута большего тензора.
2. Меньший тензор копируется в эти новые оси до полного совпадения с формой большего тензора.

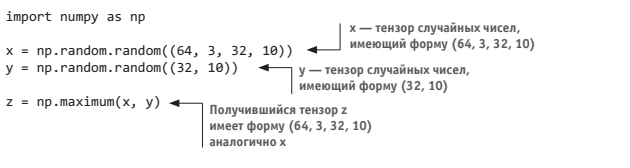
Рассмотрим конкретный пример. Пусть имеются тензоры X с формой (32, 10) и y с формой (10,). Чтобы привести их в соответствие, сначала нужно добавить в тензор y первую пустую ось, чтобы он приобрел форму (1, 10), а затем скопировать вторую ось 32 раза, чтобы в результате получился тензор Y с формой (32, 10), где Y[i, :] == y для i в диапазоне range(0, 32). После этого можно сложить X и Y, которые имеют одинаковую форму.

В фактической реализации новый двумерный тензор, конечно же, не создается, потому что это было бы неэффективно. Операция копирования выполняется чисто виртуально: она происходит на алгоритмическом уровне, а не в памяти. Но такое представление с копированием вектора для новой оси является полезной мысленной моделью. Вот как могла бы выглядеть наивная реализация:



Прием расширения в общем случае можно применять в поэлементных операциях с двумя тензорами, если один тензор имеет форму (a, b, ... n, n + 1, ... m), а другой — форму (n, n + 1, ... m). В этом случае при расширении будут добавлены оси до n - 1.

Следующий пример применяет поэлементную операцию maximum к двум тензорам с разными формами посредством расширения:



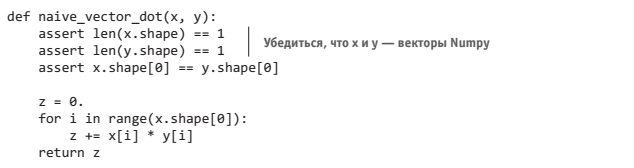
## 2.3 Скалярное произведение тензоров

Скалярное произведение, также иногда называемое тензорным произведением (не путайте с поэлементным произведением), — наиболее общая и наиболее полезная операция с тензорами. В отличие от поэлементных операций, она объединяет элементы из исходных тензоров.

Поэлементное произведение в Numpy, Keras, Theano и TensorFlow выполняется с помощью оператора \*. Операция скалярного произведения в TensorFlow имеет иной синтаксис, но в Numpy и Keras используется простой оператор dot:



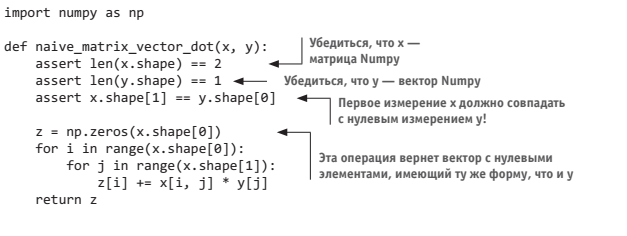
Скалярное произведение двух векторов, x и y:



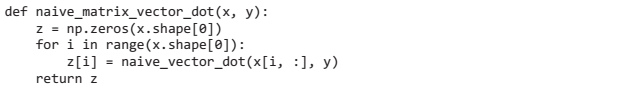
Обратите внимание, что в результате скалярного произведения двух векторов получается скаляр и в операции могут участвовать только векторы с одинаковым количеством элементов.

Также есть возможность получить скалярное произведение матрицы x на вектор y, являющееся вектором, элементы которого — скалярные произведения строк x на y.

Вот как реализуется эта операция:

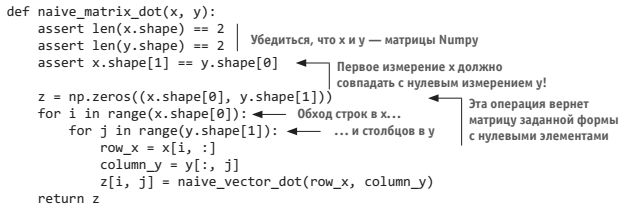


Также можно было бы повторно использовать код, написанный прежде, подчеркнув общность произведений матрицы на вектор и вектора на вектор:

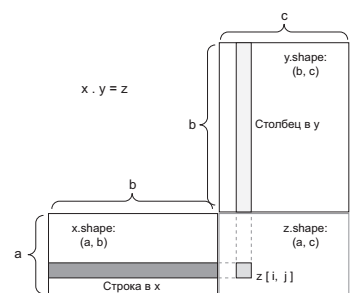


Обратите внимание, что, если один из двух тензоров имеет ndim больше 1, скалярное произведение перестает быть симметричной операцией, то есть результат dot(x, y) не совпадает с результатом dot(y, x).

Разумеется, скалярное произведение можно распространить на тензоры с произвольным количеством осей. Наиболее часто на практике применяется скалярное произведение двух матриц. Получить скалярное произведение двух матриц, x и y (dot(x, y)), можно, только если x.shape[1] == y.shape[0]. В результате получится матрица с формой (x.shape[0], y.shape[1]), элементами которой являются скалярные произведения строк x на столбцы y. Вот как могла бы выглядеть простейшая реализация:



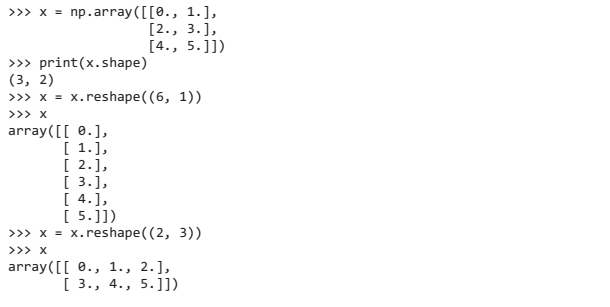
Чтобы было понятнее, как определяется совместимость форм матриц для скалярного произведения, представьте входные и выходной тензоры, как показано на рисунке.



Матрицы x, y и z изображены на рисунке в виде прямоугольников (буквально — таблицы элементов). Количество строк в x и столбцов в y должно совпадать, из чего следует, что ширина x должна совпадать с высотой y. Если вы будете создавать новые алгоритмы машинного обучения, вероятно, вам часто придется рисовать подобные диаграммы.

## 2.4 Изменение формы тензора

Изменение формы тензора предполагает такое переупорядочение строк и столбцов, чтобы привести его форму к заданной. Разумеется, тензор с измененной формой имеет такое же количество элементов, что и исходный тензор. Чтобы было понятнее, рассмотрим несколько простых примеров:



Особый случай изменения формы, который часто встречается в практике, — это транспонирование. Транспонирование — это такое преобразование матрицы, когда строки становятся столбцами, а столбцы — строками, то есть x[i, :] превращается в x[:, i]:

