JohanDDC
27 фев 2021 в 13:19

Соглашение Эйнштейна и einsum

8 мин 15K

Python*, Математика*

Из песочницы

Удивительное дело, но в русскоязычном сегменте интернета почти нет материала, разъясняющего понятным языком **соглашение Эйнштейна о суммировании.** Не менее удивительно то, что материалов, позволяющих понять принцип работы функции einsum в русскоязычном интернете ещё меньше. На английском есть довольно развёрнутый ответ о работе einsum на stack overflow, а на русском только некоторое число сайтов, предоставляющих кривой перевод этого самого ответа. Хочу исправить эту проблему с недостатком материалов, и всех, кому интересно приглашаю к прочтению!

Обсуждаем соглашение Эйнштейна

Прежде всего отмечу, что соглашение Эйнштейна чаще всего используется в тензорном анализе и его приложениях, поэтому дальше в статье будет несколько референсов к тензорам.
Когда вы только начинаете работать с тензорами, вас может смутить, что кроме привычных подстрочных индексов, используются также и надстрочные индексы, которые по началу вообще можно принять за возведение в степень. Пример:
"a с верхним индексом i" будет записано как a^i , а "a в квадрате с верхним индексом i" будет записываться $(a^i)^2$. Возможно, по-началу это вводит в заблуждение и кажется неудобным, но со временем можно привыкнуть.

Соглашение: далее в статье объекты вида $a_i x_i$ или $a_i x^i$ я буду называть *термами*.

О чём вообще соглашение Эйнштейна?

Соглашение Эйнштейна призвано уменьшить число знаков суммирования в выражении. Есть три простых правила, определяющие, насколько то или иное выражение корректно записано в

нотации Эйнштейна.

Правило № 1: Суммирование ведётся по всем индексам, повторяющимся дважды в одном терме.

Пример: рассмотрим выражение следующего вида:

$$\sum_{i=1}^{3} a_i x_i = a_1 x_1 + a_2 x_2 + a_3 x_3$$

С использованием соглашения Эйнштейна это выражение может быть переписано так:

$$a_i x_i \text{ или } a_i x^i$$

Таким образом мы избавляемся от знака суммы, и просто пишем единственный терм. Обратим внимание, что в этом терме индекс i повторяется дважды, а значит, в соответствие с первым правилом мы понимаем, что суммирование ведётся по индексу i, а точнее, по всем возможным значениям, которые принимает этот индекс.

Рассмотрим ещё один пример: пусть нам нужно умножить матрицу $A \in \mathbb{R}^{m \times n}$ на вектор $v \in \mathbb{R}^n$. Результатом будет являться вектор $b \in \mathbb{R}^m$. По определению:

$$b_i = \sum_{j=1}^{n} A_{ij} v_j,\ i = 1, \ldots, m$$

Соглашение Эйнштейна позволяет избавиться от знака суммы:

$$b_i = A_{ij} v_j = A_{ij} v^j$$

Заметим, что в терм индекс i входит один раз, а индекс j входит два раза, а значит, суммирование ведётся по индексу j.

Определение 1. Индекс, который входит в терм дважды, называется *фиктивным индексом.*

Определение 2. *Свободным индексом* назовём все индексы в терме, не являющие фиктивными.

Отметим, что каждый фиктивный индекс может быть заменён любым другим фиктивным индексом, при условии, что

1. Новый фиктивный индекс не входит в множество свободных индексов терма.
2. Новый фиктивный индекс принимает то же множество значений, что и старый фиктивный индекс.

Чтобы объяснить проще, рассмотрим следующий код на языке Python:

```
for i in range(M):
    for j in range(N):
        b[i] += A[i, j] * v[j]
```

Этот код кратко описывает процесс умножения матрицы на вектор, а точнее, этот пример. Здесь индекс j является фиктивным, а индекс i – свободным. Суммирование в соглашении Эйнштейна ведётся по фиктивным индексам. Имя переменной j мы можем заменить на любое другое.

Правило № 2. В каждом терме не может встречаться более двух одинаковых индексов.

Второе правило говорит нам, что мы можем написать $a_{ij}b_{ij}$, но не можем написать $a_{ii}b_{ij}$ или $a_{ij}b_{jj}$, несмотря на то, что на практике такие выражения всё же имеют смысл.
Больше примеров:

a_i^i – здесь i является фиктивным индексом, т.к. повторяется дважды;

a_i^{jj} – здесь i является свободным индексом, а j – фиктивным;

a_{ii}^{jj} – здесь и i, и j являются фиктивными индексами;

a_{ij}^{ij} – здесь и i, и j являются фиктивными индексами;

a_{ii}^{ij} – не правильно по второму правилу (индекс i входит в терм трижды);

Из примеров выше можно заключить, что когда мы считаем число вхождений индексов в терм, мы не делаем разницы между верхними и нижними индексами, и считаем их вместе. Ещё один важный пример: когда мы видим выражение следующего вида

$$a_{ij}b_i + a_{ji}b_j$$

Мы должны понимать, что это выражение записано верно, и не противоречит второму правилу. Действительно, если посчитать все вхождения индексов, то получится, что индекс i входит 3 раза, как и индекс j, но в выражении записано два терма, а не один, и если посчитать вхождение индексов в каждый терм отдельно (как того и требует второе правило), то мы увидим, что ничего не нарушается.

Правило № 3. В уравнениях, записанных с использованием соглашения Эйнштейна свободные индексы слева и свободные индексы справа должны совпадать.

Рассмотрим несколько примеров для закрепления этого правила:

$b_i = A_{ij}v_j$ – этот пример мы уже рассматривали выше, здесь i является свободным индексом левой части уравнения, и свободным индексом правой части уравнения;

$a_i = A_{ki}B_{kj}x_j + C_{ik}u_k$ – пример посложнее. Посчитаем вхождения индексов для каждого терма: в первый терм правой части k и j входят дважды, значит, они являются фиктивными индексами, i входит один раз, значит, является свободным. Во второй терм правой части k входит два раза, i – один, значит, k – фиктивный, i – свободный. В левой части индекс i входит один раз, а значит, является свободным. Итог: индекс i является свободным для обеих частей уравнения, а значит, правило 3 выполнено.

Рассмотрим так же несколько примеров, в которых третье правило не выполняется:

$x_i = A_{ij}$ – слева i является свободным индексом, но справа свободны индексы i и j;

$x_j = A_{ik}u_k$ – слева свободен индекс j, но справа свободен индекс i. Свободные индексы не совпадают;

$x_i = A_{ik} u_k + c_j$ – здесь слева свободен индекс i, а справа свободны индексы i, j;

Пример упрощения сложного выражения с помощью соглашения Эйнштейна: тензорный поезд

Пусть A – пятимерный тензор. Тогда утверждается, что он может быть представлен в следующем виде:

$$A_{i_1 i_2 i_3 i_4 i_5} = \sum_{j_4=1}^{R_4} \sum_{j_3=1}^{R_3} \sum_{j_2=1}^{R_2} \sum_{j_1=1}^{R_1} G^{(1)}_{i_1 j_1} G^{(2)}_{j_1 i_2 j_2} G^{(3)}_{j_2 i_3 j_3} G^{(4)}_{j_3 i_4 j_4} G^{(5)}_{j_4 i_5}$$

Там сейчас не очень важно, что из себя представляется каждая $G^{(k)}$, и что такое R_i. Наша задача сейчас – исключительно синтаксическая игра. Нужно упростить выражение, особо не вникая в смысл происходящего.
Прежде всего видно, что свободными индексами являются i_1, i_2, i_3, i_4, i_5, а фиктивными, соответственно индексы j_1, j_2, j_3, j_4. Расположим индексы в соседних множителях так, чтобы в первом множителе индекс, по которому идёт суммирование, стоял снизу, а во втором тот же самый индекс стоял сверху. Так же заметим, что множителями являются тензоры $G^{(k)}$, и у них в верхнем регистре уже стоит (k). Чтобы повысить читаемость, будем оборачивать множители в скобки, и только потом ставить индексы. Само же упрощённое выражение переписывается из исходного почти дословно:

$$A_{i_1 i_2 i_3 i_4 i_5} = \left(G^{(1)}\right)_{i_1 j_1} \left(G^{(2)}\right)^{j_1}_{i_2 j_2} \left(G^{(3)}\right)^{j_2}_{i_3 j_3} \left(G^{(4)}\right)^{j_3}_{i_4 j_4} \left(G^{(5)}\right)^{j_4}_{i_5}$$

Ура, мы научились упрощать сложные выражения с помощью соглашения Эйнштейна!

Обсуждаем einsum

einsum это функция, присутствующая в нескольких популярных библиотеках для Python (NumPy, TensorFlow, PyTorch). Во всех библиотеках, в которых эта функция реализована, она работает одинаково (с точностью до функционала структур, определённых в конкретной библиотеке), поэтому нет смысла рассматривать один и тот же пример в разных библиотеках, достаточно рассказать про einsum в одной конкретной библиотеке. Далее в статье я буду

использовать NumPy. einsum применяет соглашение Эйнштейна о суммировании к переданным массивам. Функция принимает множество опциональных аргументов, про них лучше почитать в документации, мы же сейчас разберём, как передавать шаблон, по которому функция будет применять соглашение Эйнштейна.

Рассмотрим сразу такой пример: пусть $A \in \mathbb{R}^{3\times5}$, $B \in \mathbb{R}^{5\times2}$ – две матрицы, и мы хотим их перемножить. Результатом будет матрица $M \in \mathbb{R}^{3\times2}$, которую мы можем записать следующим образом, используя определение матричного умножения и соглашение Эйнштейна:

$$M_{ij} = \sum_{k=1}^{5} A_{ik}B_{kj} = A_{ik}B_{kj}$$

Теперь пусть мы хотим перемножить их программно. Ну, это можно довольно просто сделать с помощью трёх вложенных циклов:

```
M = np.zeros((3, 2))
for i in range(3):
    for j in range(2):
        for k in range(5):
            M[i, j] += A[i, k] * B[k, j]
```

Либо, используя функцию einsum можно написать это произведение в одну строчку:

```
M = np.einsum("ik,kj->ij", A, B)
```

Разберёмся, что за магия происходит в этой строчке. einsum принимает один обязательный аргумент: шаблон, по которому будет применено соглашение Эйнштейна. Шаблон этот выглядит так:

"{индексы, определяющие размерность первого массива},{индексы, определяющие размерность второго массива}->{индексы, определяющие размерность результирующего массива}"

Поведение шаблона einsum определяется следующими правилами:

- Если один и тот же индекс встречается слева и справа от запятой (до стрелочки), то суммирование будет вестись по этому индексу;
- Если после стрелочки ничего не написано, то суммирование произойдёт по всем встреченным осям;
- Никакой индекс не должен встречаться 3 и более раз;

Таким образом мы видим, что einsum очень естественно поддерживает понятие свободных и фиктивных индексов, а также первые два правила, которые мы вводили, пока обсуждали соглашение Эйнштейна. Кроме того, как выражение, написанное с помощью соглашения Эйнштейна, может быть развёрнуто с помощью введения знаков суммы, так и функция einsum может быть развёрнута с помощью нескольких вложенных циклов. Это может быть очень удобно на первых порах, пока не сформируется устойчивое понимание einsum.

Рассмотрим теперь некоторое количество примеров разной степени сложности, чтобы закрепить понимание einsum:

Одна einsum, чтобы править всеми

Пример 1. Сумма всех значений вектора:

```
vector = np.array([1, 2, 3, 4, 5])
result = np.einsum("i->", vector)
print(result)
```

▸ Output

Пример 2. Сумма всех значений матрицы:

```
matrix = np.array([[1, 2], [3, 4], [5, 6]])
result = np.einsum("ij->", matrix)
print(result)
```

▸ Output

Пример 3. Сумма значений по столбцам:

```
matrix = np.array([[1, 2], [3, 4], [5, 6]])
result = np.einsum("ij->j", matrix)
print(result)
```

▸ Output

Пример 4. Сумма значений по строкам:

```
matrix = np.array([[1, 2], [3, 4], [5, 6]])
result = np.einsum("ij->i", matrix)
print(result)
```

▸ Output

Пример 5. Транспонирование (я об этом не написал, но оси, по которым суммирования не произошло, мы можем возвращать в любом порядке):

```
matrix = np.array([[1, 2], [3, 4], [5, 6]])
result = np.einsum("ij->ji", matrix)
print(result)
```

▸ Output

Пример 6. Умножение матрицы на вектор:

```
matrix = np.array([[1, 2], [3, 4], [5, 6]])
vector = np.array([[1, 2]])
result = np.einsum("ij,kj->ik", matrix, vector)
print(result)
```

Заметим, что вектор имеет форму 1×2, и чтобы умножить матрицу на него по правилам, его нужно было бы транспонировать. Однако с помощью einsum мы можем задать ось, по которой будет вестись суммирование, и немного выиграть по памяти, не создавая копию уже существующего вектора.

▸ Output

Пример 7. Умножение матрицы на матрицу:

```
matrix1 = np.array([[1, 2], [3, 4], [5, 6]])
matrix2 = np.array([[1, 0], [0, 1]])
result = np.einsum("ik,kj->ij", matrix1, matrix2)
print(result)
```

▸ Output

Пример 8. Скалярное произведение векторов:

```
vector1 = np.array([[1, 2, 3]])
vector2 = np.array([[1, 1, 1]])
result = np.einsum("ik,jk->", vector1, vector2)
print(result)
```

▸ Output

Пример 9. След матрицы:

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
result = np.einsum("ii->", matrix1)
print(result)
```

▸ Output

Пример 10. Адамарово (покомпонентное) произведение:

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
result = np.einsum("ij,ij->ij", matrix1, matrix2)
print(result)
```

Это может показаться контринтутивно, но, как написано выше: если не понятно, что делает einsum – запиши через циклы:

```
result = np.zeros(matrix1.shape, dtype="int32")
for i in range(result.shape[0]):
    for j in range(result.shape[1]):
        result[i, j] += matrix1[i, j] * matrix2[i, j]
print(result)
```

▸ Output

Пример 11. Кронекерово (внешнее) произведение векторов:

```
vector1 = np.array([1, 2, 3])
vector2 = np.array([1, 0, 0])
```

```
result = np.einsum("i,j->ij", vector1, vector2)
print(result)
```

▸ Output

Пример 12. Транспонирование тензора:

```
A = np.array([[[0, 1], [1, 2], [2, 3]], [[1, 2], [2, 3], [3, 4]], [[2, 3], [3, 4], [4,
result = np.einsum("ijk->jki", A)
print(result)
```

▸ Output

Пример 13. Произведение тензора на матрицу по третьей моде:

```
A = np.array([[[0, 1], [1, 2], [2, 3]], [[1, 2], [2, 3], [3, 4]], [[2, 3], [3, 4], [4,
U = np.array([[1, 2], [2, 3]])
result = np.einsum("ijk,nk->ijn", A, U)
print(result)
```

▸ Output

Итоги

Конечно, einsum поставляет только дополнительный синтаксический сахар. Всегда можно использовать цепочки вложенных циклов, множество библиотечных функций (np.dot, np.outer, np.tensordot, np.transpose, np.cumsum и т.д.), и вообще не использовать einsum. Но если потратить время и понять, как она работает, то можно научиться писать гораздо более сжатый, и, не побоюсь этого слова, эффективный код.

Ссылки

Ролик с примерами einsum (ещё больше примеров).

Соглашение Эйнштейна (база)

Соглашение Эйнштейна (продвинутая часть)

Теги: einsum, эйнштейн, numpy, соглашение эйнштейна

Хабы: Python, Математика

11 Карма 0 Рейтинг

Ваня @JohanDDC

Студент

Подписаться

Комментарии 9

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

ssichkar
18 часов назад

Вжух и денег нет: как Binance обнулил мой счет

Простой 6 мин 24K

Из песочницы

+46 37 155

Doctor_IT
23 часа назад

«Коммутаторы — это коробки, а клиенты — котики». Как устроены выделенные сетевые инсталляции

Средний 6 мин 1.6K

Обзор

+42 12 3

wofs
5 часов назад

Как мы изобрели велосипед: адаптер USB–RS485 с выходом питания 12 В и защитой

12 мин 1.7K

+35 14 21

NickSus
18 часов назад

Астрономическая «рыбалка»

Простой 6 мин 1.7K

Из песочницы

+35 18 4

melpnz
вчера в 15:21

Как мы кнопку Enter пытались сгенерировать

Простой 4 мин 2.7K

Кейс

+33 2 10

И конфеты за меня есть будете? Опыт ЕВРАЗа в AutoML

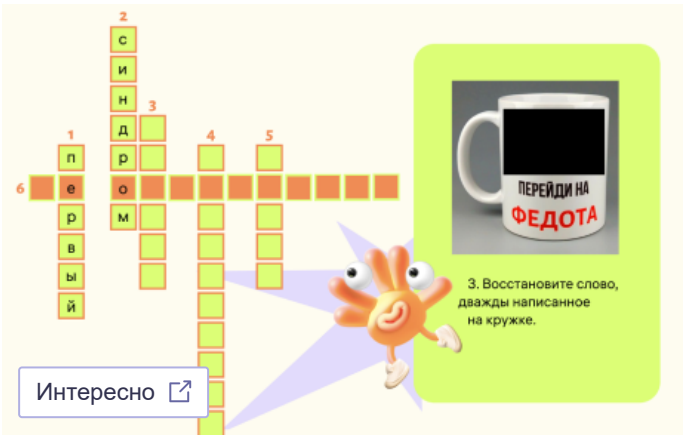
Турбо

Показать еще

МИНУТОЧКУ ВНИМАНИЯ

Разместить

Помогите джунам: напишите для них статью, а Хабр вычитает пост



В топку новомодные развлечения, решаем крестословицы!

ЗАКАЗЫ

Скрипт для amocrm

7500 руб./за проект · 1 отклик · 12 просмотров

Написать код на Python для считывания изображения и обработки картинки

8000 руб./за проект · 5 откликов · 34 просмотра

Разработка модуля на Python для внедрения в телеграм бота. Парсинг и

8000 руб./за проект · 4 отклика · 32 просмотра

Скрапинг сообщений в Телеграм каналах: python, telethon

3000 руб./за проект · 6 откликов · 40 просмотров

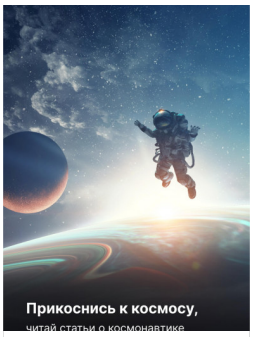
Статистическое исследование на Python/R

75000 руб./за проект · 6 откликов · 65 просмотров

Больше заказов на Хабр Фрилансе

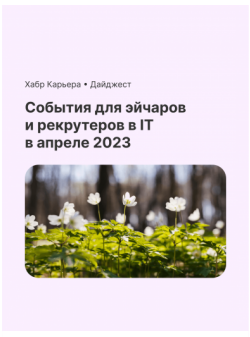
ЧИТАЮТ СЕЙЧАС

Ваш аккаунт Разделы Информация Услуги



День космонавтики

Топ-7 годноты из блогов компаний



Дайджест событий для IT HR в апреле

Сезон нейроарта на Хабре

Туториалы по Kubernetes

Ч

РАБОТА

Django разработчик
56 вакансий