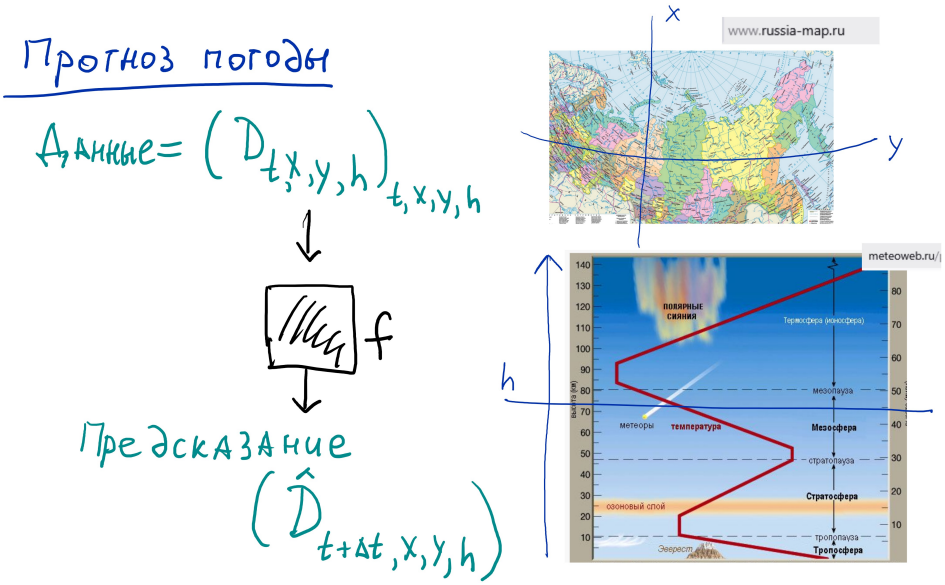
Работа с матрицами и тензорами

Наше обсуждение задач фильтрации мы начинаем со знакомства (или напоминания) основных приёмов работы с матрицами и тензорами.

1.1 Функции от матриц и тензоров

Когда дело доходит до по-настоящему реальных задач, мы замечаем, что данные – это как правило не одно число, а последовательности или векторы чисел. Функции, которые используются для постановки задач и формулировки алгоритмов их решения больше не хорошо знакомые $\mathbb{R} \to \mathbb{R}$, а некоторые процессоры, которые обрабатывают большой массив данных и возвращают на выходе большой массив данных, используя для обработки третий массив чисел (параметров).



Очень многие задачи в инженерной практике формулируются как оптимизация по набору параметров. Самый раскрученныый пример, безусловно, глубокие нейросети, которые составляются из блоков преобразований *тензоров* (под которыми мы в ходе наших обсуждений будем понимать всего-навсего многомерные таблицы, обобщение матриц, и этого нам будет достаточно). Если мы хотим применять привычную технику для подобных применений, нам нужно

- Уметь вычислять производные функций $f(X)$ от вектора (или матрицы, тензора) X.

- Эффективно реализовывать связанные с этим расчёты в своих программах.

Более того, то, что на первый взгляд не кажется как-то связанным с матрицами, может быть представлено с помощью разных матричных операций и тем самым лаконично записано в виде более простых формул.

Пример 1.1. *Пусть x_1, x_2, x_3 – три действительных числа, и a_1, a_2, a_3 – другие три действительных числа. Функция*

$$f(x, a) = a_1 x_1 + a_2 x_2 + a_3 x_3$$

может быть представлена как скалярное произведение двух векторов

$$x = [x_1, x_2, x_3]^T, \quad a = [a_1, a_2, a_3]^T$$

из $\mathbb{R}^3$ в стандартном базисе. Конкретно,

$$f(x, a) = a^T x = x^T a.$$

Формально $f : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$.

Пример 1.2. *Линейный слой (к примеру, `torch.nn.Linear`), задаётся двумя параметрами $A \in \mathbb{R}^{d_2 \times d_1}$, $b \in \mathbb{R}^{d_2}$ в виде функции*

$$f(x) = Ax + b, \quad f : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}.$$

Эта функция встречается в методе наименьших квадратов (МНК), где в качестве b и x выступают данные и требуется найти параметры A, b. Есть формула точного решения, которую можно получать разными путями.

Пример 1.3. *Свёрточный слой (к примеру, `torch.nn.Conv1d`), задаётся несколькими параметрами и позволяет задать свёртку двух векторов (детали опущены)*

$$f(x)_j = (a \star x)_j = \sum_k a_{j-k} x_k,$$

которая тоже является линейным преобразованием вектора x с матрицей очень конкретной структуры.

Нашей целью будет научиться считать производные функций $f(X)$ от векторов и матриц, которые при этом сами могут возвращать число, вектор или матрицу. На этом пути есть как хорошо знакомые вещи, так и некоторая специфика.

1.2 Соглашения и простые производные

Для начала примем несколько соглашений, которые определят то, как именно мы записываем результаты.

Мы принимаем колоночную нотацию (numerator layout). В этой нотации мы рассматриваем вектор x как вектор-колонку, а умножение матрицы для таких векторов записываем как $Ax = y$, где y тоже колонка. Особенно важно, что в такой нотации матрица Якоби ∇f функции $f : \mathbb{R}^n \to \mathbb{R}^m$ записывается как

$$\nabla f(x) = \begin{bmatrix} \partial_{x_1} f_1(x) & \partial_{x_2} f_1(x) ... \partial_{x_n} f_1(x) \\ \partial_{x_1} f_2(x) & \partial_{x_2} f_2(x) ... \partial_{x_n} f_2(x) \\ ... & \\ \partial_{x_1} f_m(x) & \partial_{x_2} f_m(x) ... \partial_{x_n} f_m(x) \end{bmatrix}$$

и в частности градиент ∇f функции $f : \mathbb{R}^n \to \mathbb{R}$ является вектором-строкой

$$\nabla f = \begin{bmatrix} \partial_{x_1} f(x) & \partial_{x_2} f(x) ... \partial_{x_n} f(x) \end{bmatrix}.$$

То есть мы выписываем размерности тензора производных (да, необязательно матрица) в следующем порядке: по числителю-столбцу (размерность числителя идёт по изменяющемуся номеру строки) и по знаменателю-строке (размерность знаменателя идёт по изменяющемся номеру столбца)

$$\frac{\partial X}{\partial Y^T}.$$

Данная нотация лучше совместима с привычным матанализом и позволяет красиво и без размышлений выписывать ряд Тейлора например. В другой нотации возникнет транспонирование.

$$f(X_0 + h) = f(X_0) + \nabla f \Big|_{X = X_0} h + \overline{o}(\|h\|)$$

Впрочем, чтобы не писать много транспонирований, мы иногда будем использовать и другое соглашение, строчную нотацию (denominator layout), что кратко записывается как

$$\frac{\partial X}{\partial Y}.$$

В этом случае количество столбцов будет определяться размерностями числителя, а количество строк – знаменателя. В случае более общих тензоров с большим числом размерностей обычно рассматривают своё соглашение, фиксируя выходные размерности (их набор в numpy называется `shape`).

В том числе в семинарской практике мы будем стараться привыкнуть к суммированию Эйнштейна, потому что это то, что нам поможет писать вычисления свёрток тензоров и других сумм очень простым образом. В суммировании Эйнштейна выделяют свободные индексы (которые встречаются один раз) и суммируемые (встречаются два раза и более). По суммируемым индексам происходит суммирование, а свободные остаются неизменными. Далее мы полагаем, что если в индексах суммы не указаны пределы, то происходит суммирование по всем возможным значениям индексов.

Пример 1.4. *Скалярное произведение векторов $x, y \in \mathbb{R}^n$ записывается как*

$$x^T y = \sum_{j=1}^{n} x_j y_j = \sum_j x_j y_j.$$

Используя соглашения Эйнштейна, мы перепишем это в виде

$$x^T y = x_i y_i.$$

Пример 1.5. *Умножение матрицы $A \in \mathbb{R}^{n \times m}$ на вектор $x \in \mathbb{R}^m$ справа запишется так:*

$$Ax = (\sum_j a_{ij} x_j)_i = a_{ij} x_j.$$

В случае, когда мы работаем с тензорами и размерностей становится больше, применяются те же правила. Такая запись не фиксирует (!) порядок выходных размерностей и о них нужно договариваться. Например, можно выстроить индексы по фиксированному порядку (обычно мы будем делать так). Для операций суммирования и произведения с соглашением Эйнштейна есть специальные функции в Numpy и Torch (`numpy/torch.einsum` и `numpy/torch.einprod`), которые чётко фиксируют размерности выхода и могут за вас транспонировать результат. Использование таких функций гораздо удобнее в плане написания кода и в каких-то аспектах может быть эффективнее по времени и памяти, чем руками делать бродкастинг и использовать стандартное матричное произведение (в Python это операция @).

Важно понять, что правила матричного дифференцирования – всего лишь удобный способ записи (хотя и тут можно углубиться в то, что такое эти производные с точки зрения операторов), поэтому многие привычные факты естественно транслируются на общий случай. Чтобы избежать тонких моментов мы будем в основном рассматривать производные скаляра по вектору или матрице, нам не понадобится больше.

1. Производная константы равна нулю. Но есть нюанс:

$$\frac{\partial a}{\partial X} = 0$$

и при этом ноль тоже имеет размерность . Например $0 \in \mathbb{R}^{n \times d}$ в случае $a \in \mathbb{R}^n$ и $X \in \mathbb{R}^d$. В случае, когда X – это матрица, размерности производной будут ещё зависеть от дополнительных соглашений.

2. Линейность работает как обычно (если производные корректно определены):

$$\frac{\partial(\alpha f + g)}{\partial X} = \alpha \frac{\partial f}{\partial X} + \frac{\partial g}{\partial X}, \quad \alpha \in \mathbb{R}.$$

Пример 1.6. *(Производная линейной функции) Рассмотрим $x \in \mathbb{R}^n$ и функцию*

$$f(A) = Ax, \quad f : \mathbb{R}^{m \times n} \to \mathbb{R}^m, \quad A \in \mathbb{R}^{m \times n}.$$

Её производная по A

$$\frac{\partial f}{\partial A} = (??)_{k,i,j},$$

где справа мы указали порядок размерностей. Некоторый тензор с тремя размерностями. Посмотрим внимательнее:

$$\frac{\partial f(A)_k}{\partial A_{ij}} = \frac{\partial f_k}{\partial A_{ij}} = \frac{\partial (A_{ks} x_s)}{\partial A_{ij}}.$$

Это уже привычная нам производная. Её значение сильно зависит от соотношения индексов матрицы в числителе и знаменателе. Заметим, что по второму индексу происходит суммирование и найдётся $s = j$. Первый индекс фиксирован, поэтому произвзодная будет ненулевая только при $i = k$. В итоге получим

$$\frac{\partial f}{\partial A} = \frac{\partial f_k}{\partial A_{ij}} = \begin{cases} 0, & i \neq k \\ x_j, & i = k. \end{cases}$$

Если же аналогичным образом посчитать производную по x, то придём к привычному

$$\frac{\partial f}{\partial x} = (??)_{k,i} = \frac{\partial (A_{ks} x_s)}{\partial x_i} = A_{ki} = A.$$

В нашем основном соглашении (numerator)

$$\frac{\partial f}{\partial A^T} = \begin{cases} 0, & j \neq k \\ x_i, & j = k, \end{cases}$$

А результат производной по x транспонируется.

Разные нотации призваны решить проблемы удобности тех или иных записей, исторически сложилось, что у разных авторов это делается по-разному и с этим приходится жить. В рамках функций $f : \mathbb{R}^n \to \mathbb{R}^m$ колоночная нотация (numerator) косметически более совместима с классическими математическими результатами (особенно это заметно в геометрии и общем понятии производной) и матричными произведениями. Но чтобы избежать лишних транспонирований при расчётах, во многих учебниках можно встретить и другую нотацию. По результату они отличаются лишь транспонированием.

Упражнение 1.1. *В более общий случай мы не идём, но вот есть простой пример на размышление. Проверьте, чему будет равна производная $f(X) = AX$ по матрице X, где A – произвольная матрица, чтобы при этом умножение было корректным. У вас получится тензор с 4 размерностями. Посчитайте*

$$\frac{\partial f}{\partial X},$$

зафиксировав размерность как вам удобно.

1.3 Формула Лейбница

Разумеется, очень хочется получить аналоги формул Лейбница и дифференцирования композиции функций. Но как мы видели ранее, нужно следить за руками, но ещё больше – за размерностями.

Положим, что у нас для простоты (посмотрите сами, что будет в более общем случае) есть две функции $f(X), g(X)$ которые в качестве результата и входа имеют матрицу и при этом произведение $f(X)g(X)$ корректно определено в матричном смысле. Как будет выглядеть производная?

$$\frac{\partial \left(f(X)g(X)\right)}{\partial X} = \frac{\partial \left(f(X)_{ij}g(X)_{jk}\right)}{\partial X_{ts}} = (??)_{ikts}.$$

Опустим вопрос четырёх размерностей (мы их расставили как захотели) и посмотрим внимательно на саму отдельную ячейку нового тензора:

$$\frac{\partial \left(f(X)_{ij}g(X)_{jk}\right)}{\partial X_{ts}}.$$

Наверху стоит функция, которая возвращает скаляр, мы берём её частную производную по входу. Обычная ситуация – тут верна формула Лейбница:

$$\frac{\partial \left(f(X)_{ij}g(X)_{jk}\right)}{\partial X_{ts}} = \frac{\partial f(X)_{ij}}{\partial X_{ts}}g(X)_{jk} + f(X)_{ij}\frac{\partial g(X)_{jk}}{\partial X_{ts}}.$$

В итоге получаем, что и тут работает формула Лейбница, но обратите внимание, как необычно происходит суммирование. Более того, заранее нужно договориться о том, в каком порядке выписывать индексы в результате.

Если же мы дифференцируем скаляр то всё проще.

Утверждение 1.1. *(Формула Лейбница, вариант) Пусть $f, g : \mathbb{R}^{m\times n} \to \mathbb{R}$, тогда*

$$\frac{\partial(f(X)g(X))}{\partial X} = \frac{\partial f(X)}{\partial X}g(X) + f(X)\frac{\partial(g(X))}{\partial X}.$$

Пример 1.7. *Рассмотрим функцию $f(X) = X^TAX$, где A, X – это матрицы с подходящими для корректности размерами. Вычислим производную:*

$$\frac{\partial\left(X^TAX\right)}{\partial X} = \frac{\partial\left(X_{ij}A_{ik}X_{kz}\right)}{\partial X_{st}} = \frac{\partial\left(X_{ij}A_{ik}X_{kz}\right)}{\partial X_{st}}.$$

Продолжаем с формулой Лейбница:

$$= \frac{\partial X_{ij}}{\partial X_{st}}A_{ik}X_{kz} + X_{ij}A_{ik}\frac{\partial X_{kz}}{\partial X_{st}}.$$

Возможны варианты. Фиксируем порядок свободных индексов: начинаем с j, z, далее s, t.

- *Это равно нулю в случае, когда $j \neq t$ и $z \neq t$.*
- *Если $j = t$ и $z \neq t$, то результат равен*

$$A_{sk}X_{kz} = (X^TA^T)_{zs}.$$

- *Если $j \neq t$ и $z = t$, то результат равен*

$$X_{ij}A_{is} = (X^TA)_{js}.$$

- *Если $j = z = t$, то результат надо очень аккуратно вытащить. В результате получим*

$$A_{sk}X_{kz} + X_{ij}A_{is} = A_{sk}X_{kz} + X_{kj}A_{ks},$$

где мы не забыли про суммирование Эйнштейна по индексу i, k. После подстановки $j = z = t$ получаем

$$= A_{sk}X_{kt} + X_{kt}(A^T)_{sk}.$$

Мораль: дополнительные размерности создают сложности, но через них можно красиво пройти.

Чем меньше размерностей, тем проще, что красиво демонстрирует следующий пример. Оператор следа

$$tr(A_{i_1,..,i_k}) = \sum_i A_{i,i,..,i}$$

определяется привычно как минимум для тензоров с одинаковыми размерностями и в результате выдаёт число. Через след, как мы увидим чуть позже, можно очень легко дифференцировать векторные нормы из методов типа наименьших квадратов.

Пример 1.8. *Рассмотрим $f(X) = tr(AX)$, где A, X – матрицы подходящего размера, чтобы было определено матричное произведение. Выпишем*

$$\frac{\partial f}{\partial X^T} = \frac{\partial f}{\partial X_{ts}^T} = \frac{\partial f}{\partial X_{st}}.$$

Пользуемся суммированием Эйнштейна, чтобы представить след:

$$= \frac{\partial \left(A_{ij} X_{ji}\right)}{\partial X_{st}}.$$

Из всех слагаемых только одно даст ненулевой результат при дифференцировании, таким образом,

$$\frac{\partial f}{\partial X_{ts}^T} = A_{ts} = A.$$

Если мы выберем строчную нотацию (denominator), то добавится транспонирование.

1.4 Цепное дифференцирование

Что касается формулы цепного дифференцирования, то здесь простой формулы нет и главная проблема в определении порядка размерностей. Но для случая, который будет очень важным для нас, мы можем получить подобную формулу.

Утверждение 1.2. *Пусть $g : \mathbb{R}^{m\times n} \to \mathbb{R}$ и функция $f : \mathbb{R} \to \mathbb{R}$. Тогда если производные корректно определены, то*

$$\frac{\partial f(g(X))}{\partial X} = f'(g(X))\frac{\partial g(X)}{\partial X}$$

и то же верно в другой нотации:

$$\frac{\partial f(g(X))}{\partial X^T} = f'(g(X))\frac{\partial g(X)}{\partial X^T}$$

Предоставим вам это проверить, это следует из привычного правила дифференцирования композиции.

Упражнение 1.2. *В методе максимальноro правдоподобия для гауссовских векторов возникает такое:*

$$f(\Sigma) = \ln\det(\Sigma).$$

Для нахождения оценок нужно вычислить $\partial f/\partial\Sigma$. Это выглядит страшно, но на самом деле очень просто. Определитель на бумаге часто считают с помощью разложения по строке или столбцу:

$$\det(\Sigma) = \sum_{i=1}^{n}(-1)^{i+k}\sigma_{ik}\Sigma_{i,k},$$

где σ_{ik} – элемент матрицы Σ, а $\Sigma_{i,k}$ обозначает минор. Минор в своём разложении не содержит σ_{ik}, что после дифференцирования останется только одно слагаемое. Заметьте, что индекс k можно менять как угодно, равно как и выбирать между разложением по строке или столбцу. Так получаем

$$\frac{\partial \det(\Sigma)}{\partial \sigma_{ts}^T} = (-1)^{s+t}\Sigma_{s,t}.$$

Это алгебраическое дополнение – не очень приятный объект, но мы можем записать результат через обратную матрицу:

$$\frac{\partial \det(\Sigma)}{\partial \Sigma^T} = \det(\Sigma)(\Sigma^{-1})^T.$$

Он следует из формулы для обратной матрицы

$$A^{-1} = \frac{1}{det(A)} A^*,$$

где A^* – это присоединённая матрица (или транспонированная матрица алгебраических дополнений).

Завершите расчёт, используя формулу цепного дифференцирования.

Упражнение 1.3. *Найдите производную по матрице X от выражений*

$$f(X) = tr(X^T A X).$$

Попробуйте напрямую посчитать результат для двух соглашений и убедиться, что у вас сошлось как ожидается. Это очень похоже на то, что мы считали до этого, но сильно проще за счёт суммирования по следу.

Литература

[1] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[2] James Durbin and Siem Jan Koopman. *Time Series Analysis by State Space Methods*, volume None of *OUP Catalogue*. Oxford University Press, 2 edition, None 2012.

[3] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[4] S.J. Julier, J.K. Uhlmann, and H.F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proceedings of 1995 American Control Conference - ACC'95*, volume 3, pages 1628–1632 vol.3, 1995.

[5] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.

[6] Siem Jan Koopman and Kai Ming Lee. Seasonality with trend and cycle interactions in unobserved components models. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 58(4):427–448, 2009.

[7] Herbert E Rauch, F Tung, and Charlotte T Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA journal*, 3(8):1445–1450, 1965.

[8] O. A. Stepanov. Kalman filtering: Past and present. an outlook from russia. (on the occasion of the 80th birthday of rudolf emil kalman). *Gyroscopy and Navigation*, 2(2):99–110, Apr 2011.

[9] R. L. Stratonovich. Conditional markov processes. *Theory of Probability & Its Applications*, 5(2):156–178, 1960.

[10] Р.Л. Стратонович. *Условные марковские процессы и их применение к теории оптимальноо управления*. Московский государственный университет, 1966.