The Tensor Cookbook

Thomas Dybdahl Ahle

July 5, 2026

Chapter 1

Introduction

What is this? These pages are a guide to tensors, using the visual language of "tensor diagrams". For illustrating the generality of the approach, I've tried to closely follow the legendary "Matrix Cookbook". As such, most of the presentation is a collection of facts (identities, approximations, inequalities, relations, ...) about tensors and matters relating to them. You won't find many results not in the original cookbook, but hopefully the diagrams will give you a new way to understand and appreciate them.

It's ongoing: The Matrix Cookbook is a long book, and not all the sections are equally amenable to diagrams. Hence I've opted to skip certain sections and shorten others. Perhaps in the future, I, or others, will expand the coverage further.

For example, while we cover all of the results on Expectation of Linear Combinations and Gaussian moments, we skip the section on general multi-variate distributions. I have also had to rearrange the material a bit, to avoid having to introduce all the notation up front.

Complex Matrices and Covariance Tensor diagrams (or networks) are currently most often seen in Quantum Physics, but this is not a book for physicists. The Matrix Cookbook is a book for engineers, in particular in Machine Learning, where complex numbers are less common. Without complex numbers, we don't have to worry about complex conjugation, which simplifies transposes, and gets rid of the need for co- and contra-variant tensors. If you are a physicist, you probably want a book on Tensor Analysis.

Tensorgrad The symbolic nature of tensor diagrams makes them well suited for symbolic computation.

Advantages of Tensor Diagram Notation: Tensor diagram notation has many benefits compared to other notations:

Various operations, such as a trace, tensor product, or tensor contraction can be expressed simply without extra notation. Names of indices and tensors can often be omitted. This saves time and lightens the notation, and is especially useful for internal indices which exist mainly to be summed over. The order of the tensor resulting from

a complicated network of contractions can be determined by inspection: it is just the number of unpaired lines. For example, a tensor network with all lines joined, no matter how complicated, must result in a scalar.

Etymology The term "tensor" is rooted in the Latin word tensio, meaning "tension" or "stretching," derived from the verb tendere, which means "to stretch" or "to extend." It was first introduced in the context of mathematics in the mid-19th century by William Rowan Hamilton in his work on quaternions, where it referred to the magnitude of a quaternion. The modern usage of "tensor" was later established by Gregorio Ricci-Curbastro and Tullio Levi-Civita in their development of tensor calculus, a framework that generalizes the concept of scalars, vectors, and matrices to more complex, multidimensional entities [2, 12].

Contents

1 Introduction **1**
- 1.1 Tensor Diagrams . . . 6
- 1.2 The Copy Tensor . . . 7
- 1.3 Sums of Tensors . . . 8
- 1.4 Transposition . . . 9
  - 1.4.1 Higher order . . . 9
- 1.5 Trace . . . 10
  - 1.5.1 Higher order traces . . . 10
- 1.6 Eigenvalues . . . 10
- 1.7 Symmetry and Symmetrization . . . 11
- 1.8 Covariance and Contravariance . . . 11
- 1.9 Exercises . . . 12

2 Simple Derivatives **13**
- 2.1 Derivatives of Matrices, Vectors and Scalar Forms . . . 14
  - 2.1.1 First Order . . . 14
  - 2.1.2 Second Order . . . 14
  - 2.1.3 Higher Order . . . 15
- 2.2 Derivatives of Traces . . . 15
  - 2.2.1 First Order . . . 16
  - 2.2.2 Second Order . . . 16
  - 2.2.3 Higher Order . . . 17
- 2.3 Exercises . . . 17

3 Kronecker and Vec Operator **19**
- 3.1 Flattening . . . 19
- 3.2 The Kronecker Product . . . 20
- 3.3 The Vec Operator . . . 22
- 3.4 Kronecker Vector Product . . . 22
- 3.5 General Matrification . . . 23
  - 3.5.1 The Lyapunov Equation . . . 23
  - 3.5.2 Encapsulating Sum . . . 23
- 3.6 The Hadamard Product . . . 24
- 3.7 Khatri–Rao product . . . 24
  - 3.7.1 Concatenation . . . 25

3.8 Derivatives . . . 26
3.9 Exercises . . . 26

4 Functions **28**
4.1 The Chain Rule . . . 30
4.1.1 The Hessian Chain Rule . . . 30
4.1.2 Chain Rule with Broadcasting . . . 31
4.1.3 Functions with multiple inputs . . . 31
4.2 Examples . . . 32
4.2.1 Known derivatives . . . 32
4.2.2 Pseudo-linear forms . . . 32
4.2.3 Trace identity . . . 33
4.2.4 Taylor . . . 33
4.3 Exercises . . . 34

5 Statistics and Probability **38**
5.1 Definition of Moments . . . 38
5.1.1 Expectation of Linear Combinations . . . 38
5.1.2 Linear Forms . . . 39
5.1.3 Quadratic Forms . . . 39
5.1.4 Cubic Forms . . . 41
5.2 Cumulants . . . 42
5.2.1 Quartic Forms . . . 43
5.3 Weighted Scalar Variable . . . 43
5.4 Gaussian Moments . . . 44
5.4.1 Gaussian Integration by Parts . . . 44
5.4.2 Quadratic and Quartic Forms . . . 44
5.4.3 Mixture of Gaussians . . . 45
5.4.4 Derivatives of the Gaussian Log-Likelihood . . . 45
5.5 Free Probability for Random Matrices . . . 46
5.5.1 Non-Crossing Partitions . . . 46
5.5.2 Free Moments and Free Cumulants . . . 46
5.5.3 R-Transform and Free Convolution . . . 47
5.5.4 Two Canonical Distributions . . . 47
5.5.5 A Note on Higher-Rank Tensors . . . 47
5.6 Exercises . . . 47

6 Determinant and Inverses **49**
6.1 Determinant . . . 49
6.2 Inverses . . . 50

7 Advanced Derivatives **51**
7.1 Derivatives of an Inverse . . . 51
7.2 Derivatives of a Determinant . . . 52
7.2.1 The Characteristic Polynomial . . . 53
7.3 Derivatives of Eigenvalues . . . 53
7.4 Derivatives of Norms . . . 53
7.5 Derivatives of Structured Matrices . . . 54

7.6 Exercises . . . 55

8 Special Tensors **56**
8.1 Fast Kronecker Multiplication . . . 56
8.2 Hadamard Transform . . . 56
8.3 Fourier Transform . . . 57
8.3.1 Multi-dimensional Fourier Transform . . . 59
8.3.2 Vandermonde Matrices . . . 59
8.4 Block matrices . . . 60
8.5 Symmetric, Skew-symmetric/Antisymmetric . . . 60
8.6 Toeplitz Matrices . . . 61
8.7 Units, Permutation and Shift . . . 61

9 Decompositions **62**
9.1 Higher-order singular value decomposition . . . 62
9.2 Rank Decomposition . . . 62
9.2.1 Border Rank . . . 62
9.3 Fast Matrix Multiplication . . . 63
9.3.1 Other . . . 64

10 Tensor Trains **65**
10.1 Cheap Operations on Trains . . . 66
10.2 What are the TT-ranks? . . . 67
10.3 TT-SVD: Computing the Decomposition . . . 68
10.4 Quantized Tensor Trains . . . 68
10.5 Sketching Tensor Products . . . 69
10.6 Exercises . . . 70

11 Machine Learning Applications **72**
11.1 Least Squares . . . 72
11.2 Hessian of Cross Entropy Loss . . . 72
11.3 Convolutional Neural Networks . . . 72
11.4 Transformers / Attention . . . 72
11.5 Tensor Sketch . . . 72
11.6 Reinforcement Learning . . . 72

12 Tensor Algorithms **73**
12.1 Tensor Contraction Orders . . . 73
12.1.1 Algorithms . . . 74

13 Tensorgrad **75**
13.1 Representation . . . 75
13.2 Canonicalization . . . 75
13.3 Differentiation . . . 76
13.4 Simplification and Evaluation . . . 77

14 Appendix **81**

1.1 Tensor Diagrams

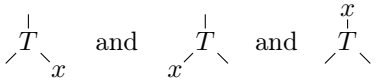
Tensor diagrams are simple graphs (or "networks") where nodes represent variables (e.g. vectors or matrices) and edges represent contractions (e.g. matrix multiplication or inner products.) The following table shows how some basic operations can be written with tensor diagrams:

Dot product	$a—b$	$y = \sum_i a_i b_i$	$[\cdot\ \cdot\ \cdot\ \cdot] \begin{bmatrix}\cdot\\\cdot\\\cdot\\\cdot\end{bmatrix} = y$
Outer product	$—a\ \ b—$	$Y_{i,j} = a_i b_j$	$\begin{bmatrix}\cdot\\\cdot\\\cdot\\\cdot\end{bmatrix} [\cdot\ \cdot\ \cdot\ \cdot] = —Y—$
Matrix-Vector	$—A–b$	$y_i = \sum_j A_{i,j} b_j$	$\begin{bmatrix}\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\end{bmatrix} \begin{bmatrix}\cdot\\\cdot\\\cdot\\\cdot\end{bmatrix} = —y$
Matrix-Matrix	$—A–B—$	$Y_{i,k} = \sum_j A_{i,j} B_{j,k}$	$\begin{bmatrix}\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\end{bmatrix} \begin{bmatrix}\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot\end{bmatrix} = —Y—$

We think of vectors and matrices as tensors of order 1 and 2. The order corresponds to the number of dimensions in their $[\cdots]$ visualization above, e.g. a vector is a 1-dimensional list of numbers, while a matrix is a 2-dimensional grid of numbers. The order also determines the degree of the node representing the variable in the tensor graph.

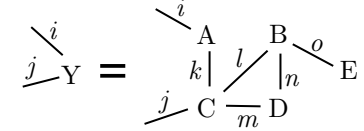
Diagram notation becomes more interesting when you have tensors of order 3 and higher. An order 3 tensor is a cube of numbers, or stack of matrices. E.g. we can write this as $T \in \mathbb{R}^{n\times m\times k}$, so $T_i \in \mathbb{R}^{m\times k}$ is a matrix for $i = 1 \ldots n$. Of course we could slice T along the other axes too, so $T_{:,j} \in \mathbb{R}^{n\times k}$ and $T_{:,:,\ell} \in \mathbb{R}^{n\times m}$ are matrices too.

A matrix having two outgoing edges means there are two ways you can multiply a vector onto it, either on the left: $x^T M$, or on the right: Mx. In graph notation we just write $x–M–$ and $–M–x$. An order 3 tensor has three edges, so we can multiply it with a vector in three ways:



To be perfectly precise about what each one means, we should give the edges labels. For example we would write  to specify the matrix $\sum_i T_i x_i$. However, often the edge in question will be clear from the context, which is part of what makes tensor diagram notation cleaner than, say, Einstein sum notation.

$$Y_{i,j} = \sum_{k,l,m,n,o} A_{i,k} B_{l,n,o} C_{j,k,l,m} D_{m,n} E_o \quad \Leftrightarrow$$



The *key principle* of tensor diagrams is that *edge contraction is associative*. This means you can contract any edge in any order you prefer. This can be seen from the sum representation above, which can be reordered to sum over k, l, m, n, o in any order.

The computational price for different contraction orders can be widely different. Unfortunately it's not computationally easy to find the optimal order. See section 12.1 for algorithms to find the best contraction order.

Note that tensor graphs are not always connected. We already saw that the outer product of two vectors can be written $-a \quad b-$. This is natural from the sum representation: No edges simply means no sums. So here $Y_{i,j} = a_i b_j$, which is exactly the outer product $Y = a \otimes b$.

1.2 The Copy Tensor

A particularly important tensor is the "copy" tensor, also known as the "diagonal", "Kronecker delta" or "spider" tensor. The simplest version is the all-ones vector, which we write as $\circ-$. That is $\circ_i = 1$. The general order-n tensor is 1 on the diagonal, 0 everywhere else:

$$\circ_{i,j,k,\ldots} = \begin{cases} 1 & \text{if } i = j = k = \ldots \\ 0 & \text{otherwise} \end{cases}$$

Or, using Iversonian notation,[1] $\circ_{i,j,k,\ldots} = [i = j = k = \ldots]$. We see the order-2 copy-tensor, $-\circ- = I$, is just the identity matrix, so we can simply remove it from graphs like this:

$$-A-\circ-B- = -A-B-$$

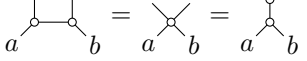
Higher order copy-tensors are very useful, because they let us turn the simple tensor graphs into hyper-graphs. A simple example of how we can use this is the diagonal matrix D_a, which has a on the diagonal and 0 elsewhere. We can write this as

$D_a =$ 

Why? Because $(D_a)_{i,j} = \sum_k \circ_{i,j,k} a_k = \sum_k [i = j = k] a_k = [i = j] a_i$. Similarly the Hadamard product, $(a \circ b)_i = a_i b_i$, can be written

$a \circ b =$ 

Now, let's see why everyone loves copy tensors by using it to prove the identity $D_a D_b = D_{a \circ b}$ by "copy tensor manipulation":

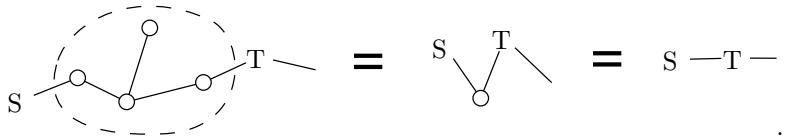
$D_a D_b =$  $=$ $=$ $= D_{a \circ b}.$

You can verify this using the sum representation.

The general rule at play is that any connected sub-graph of copy-tensors can be combined into a single one. Sometimes we are even lucky enough that this simplification

[1] For a logical proposition P, we define $[P] = \begin{cases} 1 \text{ if } P \\ 0 \text{ otherwise} \end{cases}$.

leaves us with an identity matrix we can remove too:



The only time you have to be a bit careful is when the resulting tensor has order 0. Depending on how you define the order-0 copy tensor, $\circ$, you may or may not have the identity $\circ - \circ = \circ$.

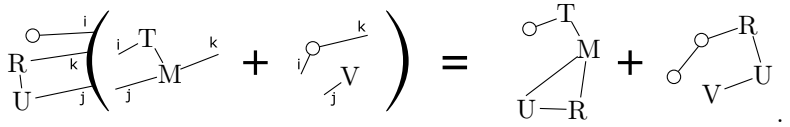
Lots of other constructions that require special notation (like diagonal matrices or Hadamard products) with normal vector notation can be unified using the copy tensor. In the Matrix Cookbook they define the order-4 tensor J, which satisfies $J_{i,j,k,l} = [i = k][j = l]$ and which we'd write as $J =$  , and satisfies, for example, $\frac{dX}{dX} = J$. Using "tensor products" you could write $J = I \otimes I$. Note that J is different from the order-4 copy-tensor,  .

1.3 Sums of Tensors

Tensor products can express any multilinear function. That is f which is linear in each argument separately; in particular $f(ax, by) = abf(x, y)$. Unfortunately not all operations on tensors are multilinear. Even something as simple as a sum of two vectors, $x + y$, cannot be displayed with a simple contraction graph. (Note that this is not bilinear because $ax + by \neq ab(x + y)$.)

To handle this important operation, Penrose suggested simply writing the two graphs with a plus sign between them, such as $-x + -y$. Note that this is itself an order-1 tensor, even though it may look like there are two free edges. If we want to multiply the sum with another tensor, we can use parentheses like $-M-(-x + -y)$.

It can be helpful to use named edges when dealing with sums, to make it clear how the edges are matched up. Sums and tensor products interact nicely, with a general form of the distributive law:



When adding tensors that don't have the same number of edges, or have edges with different names, we can use "broadcasting". Say we want to add a matrix M and a vector x. What does it even mean? If we want to add x to every row of M, we write $\frac{i}{j}M + \frac{i}{j}x$. This is because $\frac{i}{j}x$ is an outer product between x and the all-ones vector, which is a matrix in which every row is the same. Similarly, if we want to add x to every column, we could use the matrix $\frac{i}{j}x$.

Note that we typically don't talk about "rows" or "columns" when dealing with tensors, but simply use the edge names (sometimes called axes) of the tensor. When using named edges, operations from classical vector notation like "transpose" can also be removed. The

matrix X^T is simply X where the left and right edge have been swapped. But if the edges are named, we don't have to keep track of "where the edge is" at all.

1.4 Transposition

In classical matrix notation, transposition flips the indices of a matrix, so that

$$(A^T)_{ij} = A_{ji}.$$

In tensor diagram notation, we have two choices depending on whether we want the position of the edges to be significant. With significant edge positions, we typically let the "left edge" be the first index, and the "right edge" be the second index. Thus transposition requires flipping the edges:

$$(\,-A-\,)^T = \overline{\subset \underline{A} \supset} = \,-A^T-\,.$$

A fun notation used by some authors is flipping the tensor upside down, $-V-$, as a simpler way to flip the left and right edges.

In practical computations, keeping track of the edge positions can be easy to mess up. It's more robust to name the "outputs" of the tensor, and let transposition rename the edges:

$$({}_i{-}A{-}_j)^T = {}_i{-}_j A_i{-}_j$$

Renaming can be done using multiplication by identity matrices:

$$({}_i{-}A{-}_j)({}_j{-\!\circ\!-}_k) = {}_i{-}A^{j\text{-}k},$$

but we have to be careful because overlapping edge names can make multiplication non-associative. E.g.

$$\big[({}_i{-}A{-}_j)({}_j{-\!\circ\!-}_k)\big]({}_k{-\!\circ\!-}_j) = {}_i{-}A{-}_j \neq ({}_i{-}A{-}_j)\big[({}_j{-\!\circ\!-}_k)({}_k{-\!\circ\!-}_j)\big] = {}_i{-}A{-}_j \bigcirc,$$

where $\bigcirc$ equals the matrix dimension. In tensorgrad we solve this problem by requiring that any edge name is present at most twice in any product.

For the purpose of transcribing the matrix cookbook, using the "significant position" notation is more convenient. We observe the following identities:

$$(A^T)^T = A \qquad \overline{\subset \overline{\subset \underline{A} \supset} \supset} = \,-A-$$

$$(A+B)^T = A^T + B^T \qquad \overline{\subset (\,-A- \,+\, -B-\,) \supset} = \overline{\subset \underline{A} \supset} + \overline{\subset \underline{B} \supset} \tag{4}$$

$$(AB)^T = B^T A^T \qquad \overline{\subset A \underline{-B \supset}} = \overline{\subset \begin{matrix} -A \\ -B \end{matrix} \supset} = \,-B-A- \tag{5}$$

1.4.1 Higher order

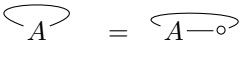
It is possible to generalize the idea of a transpose to higher order tensors. It requires partitioning the edges as "input" and "output" edges, or more commonly, "contravariant" and "covariant" edges. See the section at the end of this chapter for more on this.

We say matrices are symmetric if $A^T = A$. For higher order tensors, there are many ways to be symmetric. See the section 1.7 for more on this.

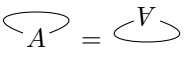
1.5 Trace

The "trace" of a square matrix is defined $\mathrm{Tr}(A) = \sum_i A_{i,i}$. In tensor diagram notation, that corresponds to a self-edge:  . The Matrix Cookbook has a list of identities using traces. Let's reproduce them with tensor diagrams:

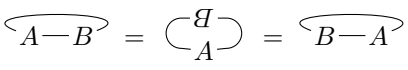
$$\sum_{i=1}^{n} A_{ii} = \mathrm{Tr}(A) = \mathrm{Tr}(AI) \qquad (11)$$



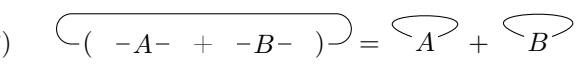
$$\mathrm{Tr}(A) = \mathrm{Tr}(A^T) \qquad (13)$$



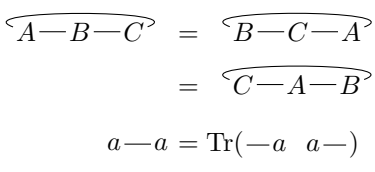
$$\mathrm{Tr}(AB) = \mathrm{Tr}(BA) \qquad (14)$$



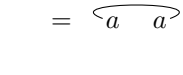
$$\mathrm{Tr}(A+B) = \mathrm{Tr}(A) + \mathrm{Tr}(B) \qquad (15)$$



$$\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA) = \mathrm{Tr}(CAB) \qquad (16)$$

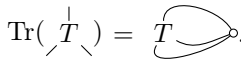


$$a^T a = \mathrm{Tr}(aa^T) \qquad (17)$$

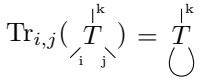


1.5.1 Higher order traces

For higher order tensors, we still define the trace as the sum of the diagonal elements. That is, for a tensor T of order n, the trace is just the product with the order n copy tensor:



In quantum mechanics, it is common to use a "partial trace", which we can define using index notation:



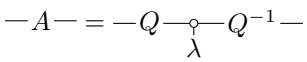
Of course with tensor diagrams, we can also use partial traces without naming the indices. We don't even have to think about whether the contractions we use are traces or not.

1.6 Eigenvalues

Eigenvalues and eigenvectors are fundamental concepts in linear algebra that have important applications in tensor network theory. In tensor diagram notation, we can represent these concepts in a visually intuitive way.

For a matrix A, if there exists a non-zero vector v and a scalar λ such that $Av = \lambda v$, then λ is called an eigenvalue of A, and v is the corresponding eigenvector. In tensor

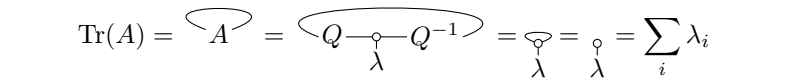
diagram notation, it's convenient to write its eigendecomposition as $A = Q\Lambda Q^{-1}$, where Q is a matrix whose columns are the eigenvectors of A, and Λ is a diagonal matrix of the eigenvalues. Thus:



(1.1)

The trace of a matrix is equal to the sum of its eigenvalues. We can represent this relationship using tensor diagrams:

$$\mathrm{Tr}(A) =$$

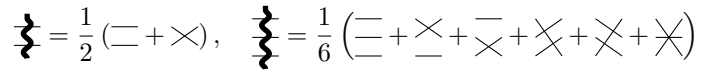


$$= \sum_i \lambda_i \tag{12}$$

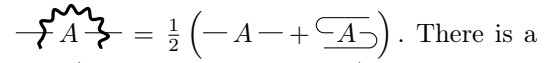
1.7 Symmetry and Symmetrization

A common property of tensors is symmetry. For matrices, symmetry means that $A^T = A$. For tensors it means that permuting its indices does not change its value. Typical examples are Covariance or Hessian matrices, but also certain higher-order statistical moments, like the third or fourth moment tensors, can exhibit symmetries. Sometimes only with respect to particular group, but most of the time with respect to all permutations of edges. The Copy Tensor is a simple example of a completely symmetric tensor.

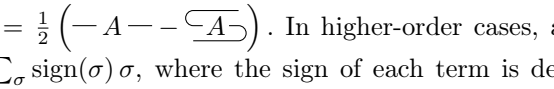
Sometimes it will be useful to symmetrize a tensor by averaging over all permutations of its indices. Following Penrose's convention, we write this using a squiggly line over the symmetrized edges. A squiggle across k edges denotes $\frac{1}{k!}\sum_\sigma \sigma$, the average over all $k!$ permutations σ of those edges:



For example, if A is a square matrix, the squiggle extracts the symmetric part $\frac{1}{2}(A + A^T)$ of A:

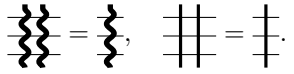


. There is a complementary notion of *anti-symmetrization* (or *skew-symmetrization*), where we average over *all permutations with appropriate sign*. For instance, an order-2 skew-symmetric matrix A satisfies $A_{ij} = -A_{ji}$. We can extract the skew-symmetric part, $\frac{1}{2}(A - A^T)$, using a flat thick line:



. In higher-order cases, a thick line across k edges denotes $\frac{1}{k!}\sum_\sigma \mathrm{sign}(\sigma)\,\sigma$, where the sign of each term is determined by the parity of the permutation.

Because of the normalization, both operators are idempotent projections: composing two of them over the same k edges gives just a single one, since symmetrizing an already-symmetric tensor has no effect:



1.8 Covariance and Contravariance

In physics it's often relevant to change between different coordinate systems. Most such changes transform the left and right side of matrices differently, and similarly row and

column vectors. For tensors this generalizes to the concepts of covariance and contravariance. With this notion, we keep track of "input" and "output" vectors. We'll also need to distinguish between things such as the identity matrix, $-\circ-$, $\succ\!\circ$, and $\circ\!\prec$.

This notion allows some ability to "algebraically" combine tensors, by connecting input edges to output edges, just like we do with matrices and vectors. However, for more complicated tensors, we need to know which edges to combine, and index notation is more useful.

In computer science and machine learning, the concept of covariance and contravariance is typically not useful, so we won't use it in this book.

1.9 Exercises

Exercise 1. Given a sequence of matrices $A_1, A_2, \ldots, A_n \in \mathbb{R}^{n\times n}$, and vectors $v_1, v_2, \ldots, v_n \in \mathbb{R}^n$, draw, using tensor diagrams, the matrix made of vectors $A_1v_1, A_2v_2, \ldots, A_nv_n$.

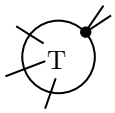
Exercise 2. Represent the Hadamard product (element-wise multiplication) of two matrices using tensor diagrams. How does this differ from regular matrix multiplication? (We will see more about this in 3.6.)

Exercise 3. Represent the SVD of a matrix $A = U\Sigma V^T$ using tensor diagrams. How does this compare to the eigendecomposition diagram? How can you generalize it to higher order tensors? (In section 9.1 we will see more about this.)

Chapter 2

Simple Derivatives

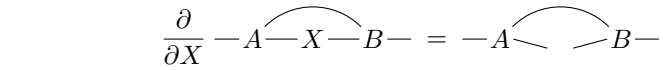
A derivative with respect to a tensor is simply the collection of derivatives with respect to each element of this tensor. We can keep track of $\frac{dT}{dU}$ by making a tensor of shape $\text{shape}(T) \cup \text{shape}(U)$. For example, if T is an order-3 tensor and U is an order-2 tensor, we draw dT/dU as

$$\frac{dT}{dU} =$$ 

This notation follows Penrose [23]. The two extra lines coming from the black dot on the circle make the derivative an order-5 tensor. That the order of derivatives grows this way, is one of the main reasons we'll encounter for tensors to show up in the first place.

When there are not too many edges, we will use a simple inline notation like this:

Before we dive into the tables, it is worth recording a simple observation that makes many of the identities below immediate. A tensor network built purely from contractions is linear in each of the tensors it contains. So if a tensor X appears exactly once in a network T, the derivative $\frac{\partial T}{\partial X}$ is just *the same network with the node X deleted*, the edges that pointed into X left dangling as the new derivative edges:

$$\frac{\partial}{\partial X}$$ 

If X appears k times, the product rule simply sums the k networks obtained by deleting one occurrence at a time — we will see this repeatedly below, e.g. in (81) where x appears twice. Note that this is *not* the chain rule: it only applies while nothing nonlinear stands between T and X. Once functions like exp or softmax enter the network, we need the machinery of Chapter 4.

The Matrix Cookbook defines the single-entry matrix $J^{i,j} \in \mathbb{R}^{n \times n}$ as the matrix which is zero everywhere except in the entry (i, j) in which it is 1. Alternatively we could write $J^{i,j}_{k,l} = [i = k][j = l]$.

2.1 Derivatives of Matrices, Vectors and Scalar Forms

2.1.1 First Order

The following first order derivatives show the basic linearity properties of the derivative operator.

$$\frac{\partial x^T a}{\partial x} = a \qquad (x — a) = (x) — a = — a \tag{69}$$

$$\frac{\partial a^T x}{\partial x} = a \qquad (a — x) = a — (x) = a_{\supset} = — a \tag{69'}$$

$$\frac{\partial a^T X b}{\partial X} = a b^T \qquad (a — X — b) = a — (X) — b = a_{\supset} {}_{\subset} b \tag{70}$$

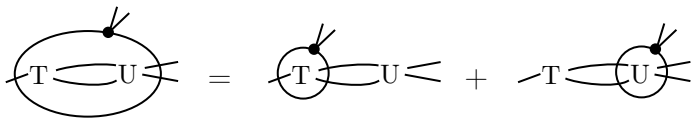
$$\frac{\partial X}{\partial X_{i,j}} = J^{i,j} \qquad (— X —)^{\mathrm{i}}_{\mathrm{j}} = {}^{\mathrm{j}}_{\mathrm{i}} \tag{73}$$

$$\frac{\partial (XA)_{i,j}}{\partial X_{m,n}} = (J^{m,n} A)_{i,j} \qquad (^{\mathrm{i}}— X — A —^{\mathrm{m}}_{\mathrm{j}})^{\mathrm{n}} = — (X) — A — \tag{74}$$

$$= {}^{\mathrm{i}}{}^{\mathrm{m}}_{\mathrm{n}} A —_{\mathrm{j}}$$

2.1.2 Second Order

The second order derivatives follow from the product rule:



Note that this rule holds independently of how many edges are between T and U, even if there are none.

$$\frac{\partial}{\partial X_{i,j}} \sum_{k,l,m,n} X_{k,l} X_{m,n} \qquad \begin{pmatrix} \circ — X — \circ \\ \circ — X — \circ \end{pmatrix}^{\mathrm{i}}_{\mathrm{j}} = \begin{matrix} \circ — X — \circ \\ \circ — (X) — \circ \end{matrix} + \begin{matrix} \circ — (X) — \circ \\ \circ — X — \circ \end{matrix} \tag{76}$$

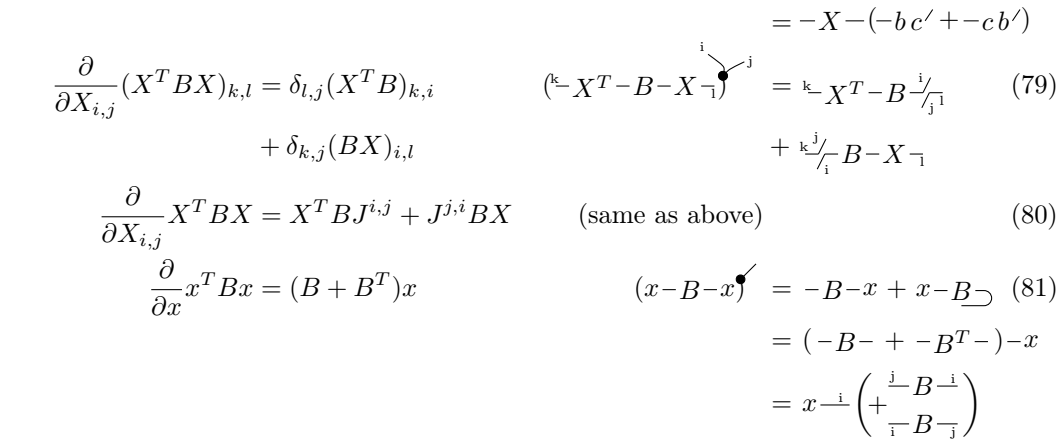
$$= 2 \sum_{k,l} X_{k,l} \qquad = 2 \begin{matrix} \circ — X — \circ \\ {}^{\mathrm{i}}\diagup_{\mathrm{j}} \end{matrix}$$

$$\frac{\partial b^T X^T X c}{\partial X} = X(b c^T + c b^T) \qquad (b — X^T — X — c) = b — X^T — (X) — c \tag{77}$$

$$+ b — (X^T) — X — c$$

$$= b — X^T —\diagup— c$$

$$+ b —\diagup— X — c$$



$$= -X-(-b\,c' + -c\,b')$$

$$\frac{\partial}{\partial X_{i,j}}(X^T B X)_{k,l} = \delta_{l,j}(X^T B)_{k,i} + \delta_{k,j}(BX)_{i,l} \tag{79}$$

$$\frac{\partial}{\partial X_{i,j}} X^T B X = X^T B J^{i,j} + J^{j,i} B X \qquad \text{(same as above)} \tag{80}$$

$$\frac{\partial}{\partial x} x^T B x = (B + B^T)x \tag{81}$$

$$= -B-x + x-B- \;+\; (-B- + -B^T-)-x$$

$$= x- \left(+ \right)$$

TODO: Assume W is symmetric, then... (84) - (88)

2.1.3 Higher Order

Integer powers of matrices, like X^n, are easy to handle by writing out the product and using the product rule. The Matrix Cookbook includes a few derivatives we can handle this way.

$$\frac{\partial (\mathbf{X}^n)_{kl}}{\partial X_{ij}} = \sum_{r=0}^{n-1} \left(\mathbf{X}^r \mathbf{J}^{ij} \mathbf{X}^{n-1-r}\right)_{kl} \tag{90}$$

$$= \quad -(X-X-X-\cdots-X)- \;+\; \cdots \;+\; -X-X-X-\cdots-(X)- \;=\; \sum_{r=0}^{n-1} -X^r - X^{n-r-1}-$$

$$\frac{\partial}{\partial \mathbf{X}} \mathbf{a}^T \mathbf{X}^n \mathbf{b} = \sum_{r=0}^{n-1} (\mathbf{X}^r)^T \mathbf{a}\mathbf{b}^T \left(\mathbf{X}^{n-1-r}\right)^T \tag{91}$$

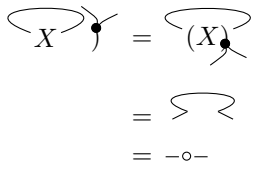
$$= \sum_{r=0}^{n-1} a - X^r \rightarrow \leftarrow X^{n-r-1} - b = \sum_{r=0}^{n-1} -(X^r)^T - a \quad b - (X^{n-r-1})^T -$$

2.2 Derivatives of Traces

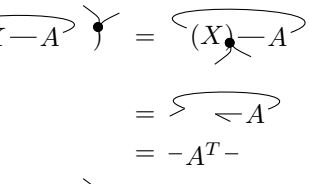
The Matrix Cookbook contains a lot of derivatives for traces. These can be elegant in classical notation, since traces are scalar, so the derivatives are low order.

2.2.1 First Order

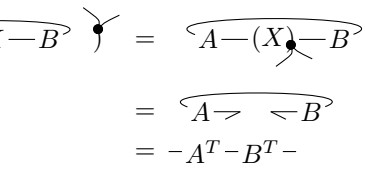
$$\frac{\partial}{\partial X}\mathrm{Tr}(X) = I \qquad (99)$$



$$\frac{\partial}{\partial X}\mathrm{Tr}(XA) = A^T \qquad (100)$$



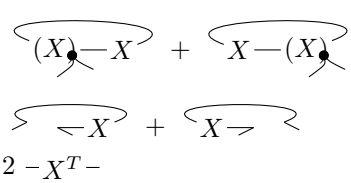
$$\frac{\partial}{\partial X}\mathrm{Tr}(AXB) = A^T B^T \qquad (101)$$



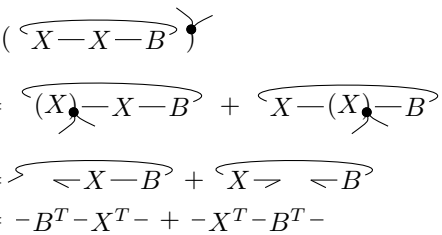
Continues for (102-105). The last one uses the Kronecker product, which we may have to introduce first.

2.2.2 Second Order

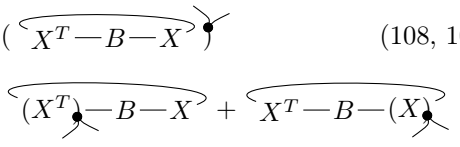
$$\frac{\partial}{\partial X}\mathrm{Tr}(X^2) = 2X^T \qquad (106)$$



$$\frac{\partial}{\partial X}\mathrm{Tr}(X^2B) = (XB + BX)^T \qquad (107)$$



$$\frac{\partial}{\partial X}\mathrm{Tr}(X^TBX) = \frac{\partial}{\partial X}\mathrm{Tr}(XX^TB) = \frac{\partial}{\partial X}\mathrm{Tr}(BXX^T) \qquad (108,\ 109,\ 110)$$



$$= (B + B^T)X \qquad = \overbrace{\prec B — X}^{} + \overbrace{X^T — B \succ}^{}$$
$$= -B-X- + -B^T-X-$$

$$\frac{\partial}{\partial X}\mathrm{Tr}(XBX^T) = \frac{\partial}{\partial X}\mathrm{Tr}(X^TXB) \qquad (\overbrace{X — B — X^T}^{}\bullet\!\!\rangle \qquad (111, 112, 113)$$
$$= \frac{\partial}{\partial X}\mathrm{Tr}(BX^TX) \qquad = \overbrace{(X\bullet\!\!\rangle — B — X^T}^{} + \overbrace{X — B — (X^T\bullet\!\!\rangle}^{}$$
$$= X(B^T + B) \qquad = \overbrace{\prec B — X^T}^{} + \overbrace{X — B \succ}^{}$$
$$= -X-B^T- + -X-B-$$

The last equation is a bit surprising, since we might assume we could simply substitute X for X^T in the previous equation and conclude

$$(B + B^T)X = \frac{\partial}{\partial X}\mathrm{Tr}(X^TBX) = \frac{\partial}{\partial X}\mathrm{Tr}(XBX^T) = X(B^T + B).$$

However that is clearly not the case. Such substitution would only work for a linear function, not a quadratic. In general one cannot push the substitution $X \to X^T$ through the derivative: $\frac{\partial}{\partial X}f(X^T)$ is not simply $\frac{\partial}{\partial X}f(X)$ with X replaced by X^T.

2.2.3 Higher Order

$$\frac{\partial}{\partial X}\mathrm{Tr}(X^n) = n(X^{n-1})^T \qquad \overbrace{(X-X-X-\cdots-X\bullet\!\!\rangle}^{} \qquad (121)$$
$$= \sum_{r=0}^{n-1} \overbrace{X^r \succ \prec X^{n-r-1}}^{}$$
$$= n(X^T)^{n-1}$$

$$\frac{\partial}{\partial X}\mathrm{Tr}(AX^n) = \sum_{r=0}^{n-1}(X^rAX^{n-1-r})^T \qquad \overbrace{(A-X-X-\cdots-X\bullet\!\!\rangle}^{} \qquad (122)$$
$$= \sum_{r=0}^{n-1} \overbrace{A-X^r \succ \prec X^{n-r-1}}^{}$$
$$= \sum_{r=0}^{n-1} -(X^r)^T-A^T-(X^{n-r-1})^T-$$

2.3 Exercises

Exercise 4. Find the derivative of $(x^TAx)^2$ with respect to x.

Exercise 5. Find the second derivative of

$$x^TA^Txx^TAx$$

with respect to x.

Exercise 6. Find the derivative of $X^T AX$ with respect to X.

Exercise 7. Show the derivatives:

$$\frac{\partial}{\partial x}(Bx+b)^T C(Dx+d) = B^T C(Dx+d) + D^T C^T (Bx+b) \tag{78}$$

$$\frac{\partial}{\partial X} b^T X^T DXc = D^T Xbc^T + DXcb^T \tag{82}$$

$$\frac{\partial}{\partial X}(Xb+c)^T D(Xb+c) = (D+D^T)(Xb+c)b^T \tag{83}$$

Exercise 8. Show the remaining second order trace derivatives from the Matrix Cookbook:

$$\frac{\partial}{\partial X}\operatorname{Tr}(AXBX) = A^T X^T B^T + B^T X^T A^T \tag{114}$$

$$\frac{\partial}{\partial X}\operatorname{Tr}\left(X^T X\right) = \frac{\partial}{\partial X}\operatorname{Tr}\left(XX^T\right) = 2X \tag{115}$$

$$\frac{\partial}{\partial X}\operatorname{Tr}\left(B^T X^T CXB\right) = C^T XBB^T + CXBB^T \tag{116}$$

$$\frac{\partial}{\partial X}\operatorname{Tr}\left[X^T BXC\right] = BXC + B^T XC^T \tag{117}$$

$$\frac{\partial}{\partial X}\operatorname{Tr}\left(AXBX^T C\right) = A^T C^T XB^T + CAXB \tag{118}$$

$$\frac{\partial}{\partial X}\operatorname{Tr}\left[(AXB+C)(AXB+C)^T\right] = 2A^T(AXB+C)B^T \tag{119}$$

Exercise 9. Show the derivative of the fourth order trace from the Matrix Cookbook:

$$\begin{aligned}\frac{\partial}{\partial X}\operatorname{Tr}\left[B^T X^T CXX^T CXB\right] = {} & CXX^T CXBB^T \\ & + C^T XBB^T X^T C^T X \\ & + CXBB^T X^T CX \\ & + C^T XX^T C^T XBB^T\end{aligned} \tag{123}$$

Chapter 3

Kronecker and Vec Operator

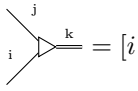
3.1 Flattening

Flattening is a common operation for programmers. In the language of numpy, we may write `np.ones((2,3,4)).reshape(2, 12)` to flatten a shape (2,3,4) tensor into a shape (2,12) matrix. Similarly, in mathematical notation, $\text{vec}(X)$ is commonly used to denote the flattening of a matrix into a vector.

Typically the main reason to do this is as a kludge for dealing with bad general notation for tensors. Hence, with tensor diagrams, we can avoid this operation entirely. However, it is still interesting to see how tensor diagrams can make a lot of properties of flattening much more transparent.

To begin with we note that flattening is a linear operation, and hence can be represented as a simple tensor. We'll use a triangle to denote this:

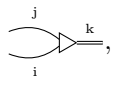
$$\triangleright_{i,j,k} = \quad = [i + jn = k].$$



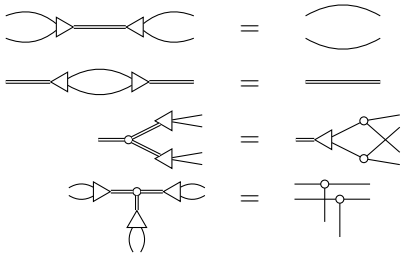
Here n is the dimension of the i edge. Note we use a double line to denote the output of the flattening operation. This is simply a syntactic choice to remind ourselves that the output is a bundle of two edges.

Which edge goes on top? Throughout the book we use the convention that *the top edge of a bundle is its most significant part*: reading a bundle top to bottom matches reading the classical expression left to right, the way we write numbers with the most significant digit first. In the definition above, the bottom edge i is the fast index and the top edge j selects the block. For the Kronecker products below, this means $A \otimes B$ is drawn with A on top. And since $\text{vec}(X)$ stacks the *columns* of X, the column index is the most significant part of its bundle: when X is drawn to the left of a flattening triangle, its column edge takes the top port and its row edge wraps under into the bottom port. This convention also agrees with the wire order of the FFT diagrams in the Special Tensors chapter, where the top wire carries the most significant bit. One caveat: bundles behave like ribbons, so a double wire that turns 180° around comes back with its wires in reversed order; when we intend the order to be preserved we mark a compensating *twist* — see the transpose rule (510) below.

Using this notation we can write

$$\mathrm{vec}(X)_k = \sum_{i,j} \triangleright_{i,j,k} X_{i,j} =$$ 

where the row index i of X takes the fast, bottom port and the column index j the top. Some basic properties of $\triangleright$:



(3.1)

(3.2)

(3.3)

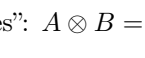
(3.4)

3.2 The Kronecker Product

The Kronecker product of an $m \times n$ matrix A and a $p \times q$ matrix B, is an $mp \times nq$ matrix, $A \otimes B$ defined as

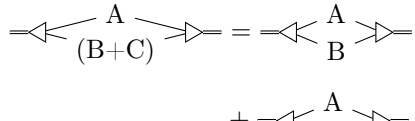
$$A \otimes B = \begin{bmatrix} A_{1,1}B & A_{1,2}B & \cdots & A_{1,n}B \\ A_{2,1}B & A_{2,2}B & \cdots & A_{2,n}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1}B & A_{m,2}B & \cdots & A_{m,n}B \end{bmatrix}.$$

Using index notation we can also write this as $(A \otimes B)_{p(i-1)+k,\, q(j-1)+l} = A_{ij}B_{kl}$, but it's pretty hard to read.

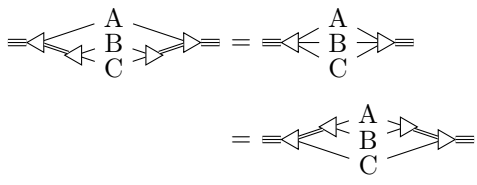
In tensor notation the Kronecker Product is simply the outer product of two matrices, flattened "on both sides": $A \otimes B =$ 

.

The Kronecker product has the following properties:

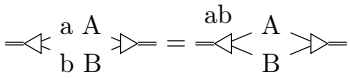
$$A \otimes (B + C) = A \otimes B + A \otimes C$$ 

(506)

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C$$ 

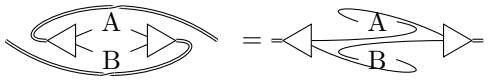
(508)

$$aA \otimes bB = ab(A \otimes B)$$



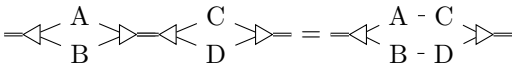
(509)

$$(A \otimes B)^T = A^T \otimes B^T$$



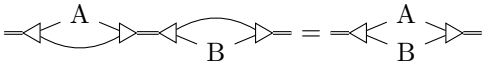
(510)

$$(A \otimes B)(C \otimes D) = AC \otimes BD$$



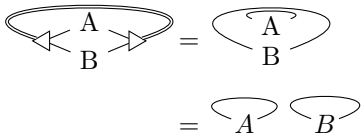
(511)

$$(A \otimes I)(I \otimes B) = A \otimes B$$



(511b)

$$\mathrm{Tr}(A \otimes B) = \mathrm{Tr}(A)\mathrm{Tr}(B)$$

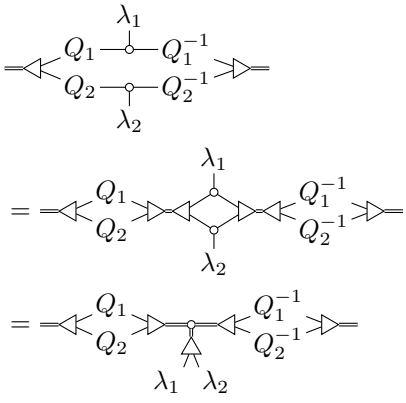


(515)

Note the small *twists* on the wrapped bundles in (510). A double wire is genuinely two parallel wires, and dragging a bundle through a 180° turn reverses their order on the page. The twist — the two wires crossing once — compensates, so the dangling bundle again reads A before B. Without the twists, the left-hand side of (510) would be the *bit-reversed* transpose $B^T \otimes A^T$, a genuinely different matrix! For the same reason, rotating a flattened block 180° is *not* its transpose. The trace loop in (515), on the other hand, needs no twist: it turns a full 360°, so the two reversals cancel. As a bonus, the twist on a straight bundle is a useful tensor in its own right: it is exactly the *commutation matrix* K, satisfying $K(x \otimes y) = y \otimes x$ and $K\,\mathrm{vec}(X) = \mathrm{vec}(X^T)$.

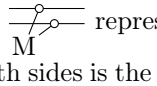
Recalling the eigen decomposition of a matrix, (1.1), we also get:

$$\mathrm{eig}(A \otimes B) = \mathrm{eig}(A)\mathrm{eig}(B)$$



(519)

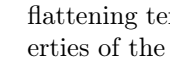
where the last step is due to (3.4).

This is easier to see when we consider that $V =$  represents the tensor where $V_{i,j,i,j} = M_{i,j}$ and 0 otherwise. So flattening V on both sides is the same as $\mathrm{diag}(\mathrm{vec}(M))$.

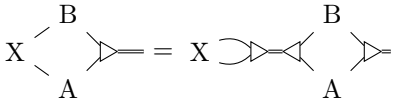
3.3 The Vec Operator

The vec-operator applied on a matrix A stacks the columns into a vector, i.e. for a 2×2 matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{vec}(A) = \begin{bmatrix} A_{11} \\ A_{21} \\ A_{12} \\ A_{22} \end{bmatrix}$$

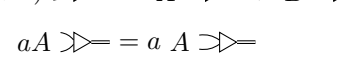
At the start of the chapter we showed how to represent the vec-operator using the flattening tensor: $\text{vec}(X) =$ 

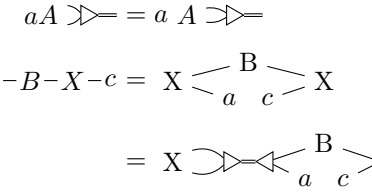
. The Matrix Cookbook gives the following properties of the vec-operator:

$$\text{vec}(A^T X B)^T = \text{vec}(X)^T (B \otimes A) \tag{520}$$


$$\text{Tr}(A^T B) = \text{vec}(A)^T \text{vec}(B) \tag{521}$$


$$\text{vec}(A + B) = \text{vec}(A) + \text{vec}(B) \tag{522}$$


$$\text{vec}(aA) = a\,\text{vec}(A) \tag{523}$$


$$a^T X B X^T c = \text{vec}(X)^T (B \otimes a c^T) \text{vec}(X) \tag{524}$$


3.4 Kronecker Vector Product

Sometimes it's convenient to write the Kronecker product between two vectors as a matrix. We define

$$A \otimes v = \tag{3.5}$$


$$A \otimes v^T = \tag{3.6}$$


$$v \otimes A = \tag{3.7}$$


$$v^T \otimes A = \tag{3.8}$$

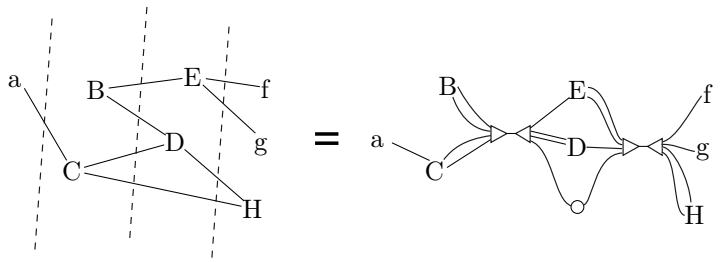

Similarly, we can define the Kronecker product between two vectors:

$$v \otimes w = \tag{3.9}$$


$$v^T \otimes w^T = \tag{3.10}$$


3.5 General Matrification

The last equation is an example of a general idea: Any tensor network can be transformed into a series of matrix multiplications by applying the vec-operator to all tensors and the flattening tensor to all edges. For example, the following complicated graph:



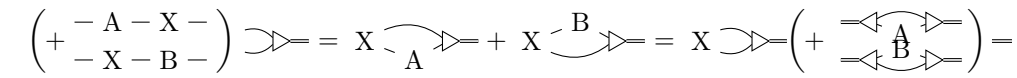
Can be written as a simple vector-matrix-matrix-vector product, $a^T M_1 M_2 b$, where $M_1 = \text{vec}(B) \otimes C'$, $M_2 = E' \otimes D' \otimes I$ and $b = f \otimes g \otimes \text{vec}(H)$, where C', D' and E' are rank 3 tensors flattened on one side, and $\text{vec}(B)$ is interpreted as a matrix with a single column.

3.5.1 The Lyapunov Equation

A nice application of Kronecker product rewritings is to solve equations like

$$AX + XB = C. \tag{272}$$

We use the rewriting $\text{vec}(AX + XB) = (I \otimes A + B^T \otimes I)\text{vec}(X)$, which follows from the tensor diagram massaging:



after which we can take the normal matrix inverse to get

$$\text{vec}(X) = (I \otimes A + B^T \otimes I)^{-1}\text{vec}(C). \tag{273}$$

3.5.2 Encapsulating Sum

This is a generalization of the previous equation.

$$\sum_n A_n X B_n = C \tag{274}$$

$$\text{vec}(X) = (\sum_n B_n^T \otimes A_n)^{-1}\text{vec}(C) \tag{275}$$

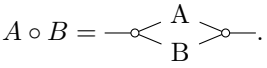
3.6 The Hadamard Product

The Hadamard product, also known as element-wise multiplication, is not described in the Matrix Cookbook. Yet, it is a very useful operation, and has some interesting properties in connection with the Kronecker product.

We define the Hadamard product of two 2×2 matrices A and B as

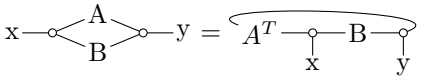
$$A \circ B = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \circ \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} \\ A_{21}B_{21} & A_{22}B_{22} \end{bmatrix}.$$

In tensor notation, the Hadamard product can be represented using two rank-3 copy tensors:

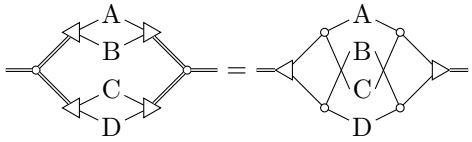
$A \circ B =$ 

Some properties of the Hadamard product are:

$$x^T(A \circ B)y = \mathrm{tr}(A^T D_x B D_y)$$



$$(A \otimes B) \circ (C \otimes D) = (A \circ C) \otimes (B \circ D)$$



The first equation is simply massaging the tensor diagram. The second follows from (3.3). Alternatively, it suffices to follow the double lines to see that A and C both use the "upper" part of the double edge, while B and D use the lower part.

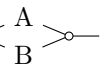
3.7 Khatri–Rao product

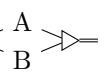
Also known as the column-wise Kronecker, row-wise Kronecker or "Face-splitting Product". We use the symbols $*$ and $\bullet$ for the column and row-wise Kronecker products, respectively.

$$A * B = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} \\ A_{11}B_{21} & A_{12}B_{22} \\ A_{21}B_{11} & A_{22}B_{12} \\ A_{21}B_{21} & A_{22}B_{22} \end{bmatrix}$$

$$A \bullet B = \ldots = \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} & A_{12}B_{11} & A_{12}B_{12} \\ A_{21}B_{21} & A_{21}B_{22} & A_{22}B_{21} & A_{22}B_{22} \end{bmatrix}$$

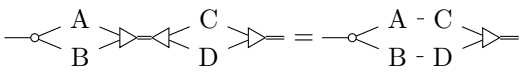
In terms of tensor diagrams, these products correspond simply to flattening the product on one side, and using a copy tensor on the other:

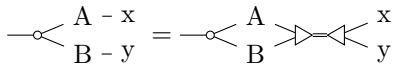
$A * B =$ 

$A \bullet B =$ 

Clearly the two are identical up to transpose. Indeed, $(A*B)^T = A^T \bullet B^T$ and $(A\bullet B)^T = A^T * B^T$.

There are multiple "mixed product" identities:

$(A \bullet B)(C \otimes D) = (AC) \bullet (BD)$ 

$(Ax) \circ (By) = (A \bullet B)(x \otimes y)$ 

3.7.1 Concatenation

If two tensors have matching shapes, we can concatenate them along an axis. An easy way to do this is taking the outer product with a vector that selects the location in the new axis, and then add them together. Define $e^{(i)}$ to be the vector of length 2 with a 1 in the i-th position and 0 elsewhere. I.e. $e^{(0)} = [1, 0]$ and $e^{(1)} = [0, 1]$. Then we can define

$$\text{stack}_i(X, Y) = \overset{i}{=\!\!<} e^{(0)} \; X \; \overline{}^{j} + \overset{i}{=\!\!<} e^{(1)} \; Y \; \overline{}^{j}.$$

For matrices and vectors, we often write concatenation using bars, like: $\text{stack}_i(A, B) = A \mid B$ and $\text{stack}_j(A, B) = \frac{A}{B}$. From this we get the identity

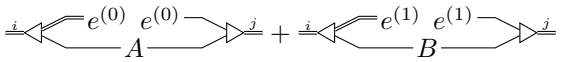
$$\begin{aligned}
(A \mid C)\begin{pmatrix} B \\ D \end{pmatrix} &= \text{stack}_i(A, C)\,\text{stack}_j(B, D) \\
&= \left(\overset{i}{=\!\!<} e^{(0)} \; A \; \overline{}^{j} + \overset{i}{=\!\!<} e^{(1)} \; C \; \overline{}^{j} \right) \left(\overset{i}{=\!\!<} e^{(0)} \; B \; \overline{}^{m} + \overset{i}{=\!\!<} e^{(1)} \; D \; \overline{}^{m} \right) \\
&= \overset{e^{(0)}-e^{(0)}}{\overline{}^{j}} A — B \; \overline{}^{m} + \overset{e^{(0)}-e^{(1)}}{\overline{}^{j}} A — D \; \overline{}^{m} + \overset{e^{(1)}-e^{(1)}}{\overline{}^{j}} C — D \; \overline{}^{m} + \overset{e^{(1)}-e^{(0)}}{\overline{}^{j}} C — B \; \overline{}^{m} \\
&= AB + CD,
\end{aligned}$$

since ${e^{(0)}}^T e^{(0)} = {e^{(1)}}^T e^{(1)} = 1$, and ${e^{(0)}}^T e^{(1)} = 0$.

The stacking operation is closely related to the direct sum of matrices. The direct sum $A \oplus B$ of two matrices creates a block diagonal matrix:

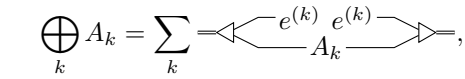
$$A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$$

In tensor diagram notation, we can express the direct sum using a construction similar to stacking:

$$A \oplus B =$$ 

The construction makes it clear that the direct sum places A in the upper-left block (when $i = j = 0$) and B in the lower-right block (when $i = j = 1$), with zeros elsewhere. This connection is particularly important in quantum computing, where this operation is known as "controlling" - applying different operations based on the state of a control qubit.

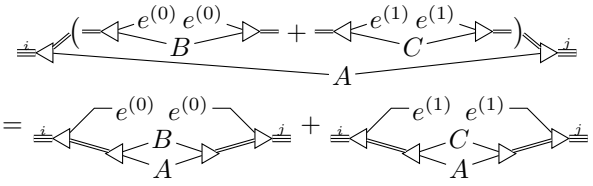
More generally, for multiple matrices:

$$\bigoplus_k A_k = \sum_k$$ 

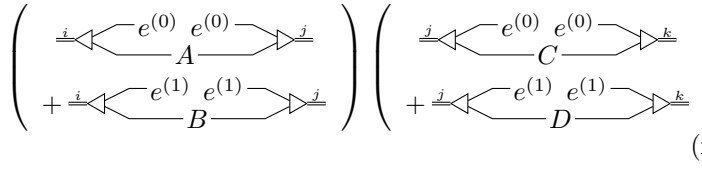
,

where we define $e^{(k)}$ as a vector of length n with a 1 in the k-th position and 0 elsewhere, for $k = 0, 1, \ldots, n-1$.

The direct sum satisfies several properties with respect to the Kronecker product:

$$(B \oplus C) \otimes A = (B \otimes A) \oplus (C \otimes A)$$ 

(yyy)

$$(A \oplus B)(C \oplus D) = AC \oplus BD$$ 

(xxx)

3.8 Derivatives

3.9 Exercises

Exercise 10. Let $J = \mathrm{Tr}[(I_N \otimes F)^T A (I_N \otimes F) B]$ where $F \in \mathbb{R}^{N \times Nn}$, $A \in \mathbb{R}^{N^2 \times N^2}$, $B \in \mathbb{R}^{N^2 n \times N^2 n}$. Find the derivative of J with respect to F.

Exercise 11. Consider $J = \|G - (B \otimes X)\|_F^2$, where G and B are matrices, and $\|\cdot\|_F$ is the Frobenius norm. Find the derivative with respect to X

Exercise 12. Prove the equation [10]:

$$\mathrm{vec}(A \,\mathrm{diag}(b)\, C) = ((C^T \otimes 1) \circ (1 \otimes A))b.$$

Here $\mathrm{diag}(b)$ is a diagonal matrix with the vector b on the diagonal, 1 is a vector of ones of the right size, and $b \otimes A$, the Kronecker product for a vector and a matrix, is defined by $=\!\!\lessdot\!\!\lessdot \overset{b}{A}\!\!-$, that is, you just flatten on one side.

Exercise 13. Let a and b be two vectors and let D and X be two matrices. Minimize the following cost function with respect to X:

$$E = \|a - DXb\|_2^2.$$

Exercise 14. Prove using diagrams that $\mathrm{Tr}(A^T B) = 1^T \mathrm{vec}(A \circ B)$, where 1 is a vector of the appropriate size.

Exercise 15.

$$\frac{\partial}{\partial \mathbf{X}} \mathrm{Tr}(\mathbf{X} \otimes \mathbf{X}) = \frac{\partial}{\partial \mathbf{X}} \mathrm{Tr}(\mathbf{X}) \mathrm{Tr}(\mathbf{X}) = 2 \mathrm{Tr}(\mathbf{X})\mathbf{I}$$

Exercise 16. Take the derivative of

$$\mathrm{Tr}(G(A \otimes X))$$

with respect to X, where $G \in \mathbb{R}^{n^2 \times n^2}$, and A and X are $\mathbb{R}^{n \times n}$ matrices.

Write the result in terms of $\mathrm{vec}(A)$ and a reshaped version of G.

Exercise 17. Verify the following identity:

$$\frac{d}{dx} \mathrm{vec}(\mathrm{diag}(x)\, A\, \mathrm{diag}(x)) = \mathrm{diag}(\mathrm{vec}(A))(I \otimes x + x \otimes I).$$

Hint: Use the matrix-vector identities (3.5).

Exercise 18. Show that

$$\frac{\partial}{\partial x} x^T A\, (x \otimes x) = A(x \otimes x) + (x^T \otimes I + I \otimes x^T) A^T x$$

Hint: Use the matrix-vector identities (3.5).

Chapter 4

Functions

In this chapter we explore general functions from tensors into other tensors. While not quite as elegant as linear functions, they are important for many applications, such as non-linearities in neural networks. The main goal is to make the vector chain rule intuitive and easy to apply, but we also cover Taylor series, determinants and other topics.

Standard notation for function over vectors can be quite ambiguous when the function broadcasts over some dimensions. If $x \in \mathbb{R}^n$ is a vector, it's not clear whether $f(x)$ applies to each element of x independently or if it's a function of the whole vector. With tensor diagrams, we make this distinction clear by explicitly showing the edges of x that go into f, and which don't. We use the notation $\overset{m}{—} f \overset{n}{\longleftarrow} x$ when f is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$. If f is element-wise, we write $(f \overset{n}{\longleftarrow} x \overset{n}{—}) \in \mathbb{R}^n$. We always assume that the function (arrow) edges are contracted before normal edges. If we want something else, like $f(Mx)$, we can use brackets: $— f \longleftarrow (- M - x)$. It may be helpful with some more examples. In the examples below, $x \in \mathbb{R}^n$:

$f : \mathbb{R}^n \to \mathbb{R}$	$f(x) \in \mathbb{R}$	$f \xleftarrow{n} x$	(Scalar function)
$g : \mathbb{R}^n \to \mathbb{R}^m$	$g(x) \in \mathbb{R}^m$	$\xrightarrow{m} g \xleftarrow{n} x$	(Vector function)
$h : \mathbb{R} \to \mathbb{R}$	$h(x) \in \mathbb{R}^n$	$h \leftarrow\!\cdots x \text{ —}^{n}$	(Element-wise function)
$u : \mathbb{R}^n \to \mathbb{R}^m$ $v \in \mathbb{R}^m$	$v^T u(x) \in \mathbb{R}$	$v \xrightarrow{m} u \xleftarrow{n} x$	(Vector times vector function)
$A : \mathbb{R}^n \to \mathbb{R}^{m \times n}$, $v \in \mathbb{R}^n$	$A(x)v \in \mathbb{R}^m$	$\xrightarrow{m} A \overset{n}{\underset{n}{\swarrow\nwarrow}} \begin{matrix} v \\ x \end{matrix}$	(Vector times matrix function)
$f : \mathbb{R}^d \to \mathbb{R}$, $X \in \mathbb{R}^{b \times d}$	$f(X) \in \mathbb{R}^b$	$f \xleftarrow{\text{d}} X \xrightarrow{\text{b}}$	(Vector function, batched)
$A : \mathbb{R}^{n \times m} \to \mathbb{R}$, $X \in \mathbb{R}^{n \times m}$	$A(X) \in \mathbb{R}$	$A \overset{\text{n}}{\underset{\text{m}}{\circlearrowleft}} X$	(Matrix input function)
$f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^d$, $u \in \mathbb{R}^{b \times n}, v \in \mathbb{R}^{b \times m}$	$f(u, v) \in \mathbb{R}^{b \times d}$	$\xrightarrow{d} f \overset{n}{\underset{m}{\swarrow\nwarrow}} \begin{matrix} u \\ v \end{matrix} \rightarrowtail\!\!\circ\!\!\text{—}^{b}$	(Two inputs, batched)

To make it more concrete, let's consider some well known functions: (1) The determinant function, $\det : \mathbb{R}^{n \times n} \to \mathbb{R}$ is a scalar function, and we can apply it to a batch of matrices by $\det \overset{\text{n}}{\underset{\text{n}}{\circlearrowleft}} X \text{ —}^{b}$ which results in a single vector $\in \mathbb{R}^b$. (2) Cosine similarity, $\text{cossim} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a scalar function, and we can apply it to two vectors by $\text{cossim} \overset{n}{\underset{n}{\swarrow\nwarrow}} \begin{matrix} u \\ v \end{matrix}$ which results in a single value $\in \mathbb{R}$. (3) If the element-wise power function, $\text{pow}_n : \mathbb{R} \to \mathbb{R}$ is applied twice to the same vector, the factors simplify under the Hadamard product:

$$\begin{matrix} \text{pow}_n \leftarrow\!\cdots u \\ \text{pow}_m \leftarrow\!\cdots u \end{matrix} \rightarrowtail\!\!\circ\!\!\text{—} = \text{pow}_{n+m} \leftarrow\!\cdots u \text{ —}$$

For a more advanced example, consider the softmax function: $\xrightarrow{d} \text{softmax} \xleftarrow{d} x$. We can write this using elementary functions as:

$$\text{softmax}(x) = \frac{\exp(x)}{\text{sum}(\exp(x))} = \begin{matrix} \exp \leftarrow\!\cdots x \text{ —} \\ \text{pow}_{-1} \leftarrow\!\cdots (\ \exp \leftarrow\!\cdots x \text{ —}\!\circ\ ) \end{matrix}$$

It looks a bit complicated at first, but let us break it down step by step: $(\ \exp \leftarrow\!\cdots x \text{ —}) \in \mathbb{R}^d$ is the element-wise exponential function. If we contract it with —○, we get the sum $s = \sum_i \exp(x_i)$. Finally, we apply pow_{-1} to s and multiply it with the exponential function to get the softmax function.

One situation where things may get a bit confusing is when the output tensor is contracted with the broadcasting edges of an input tensor. But as long as we remember to contract the function edges first, things work out. For example, for a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and a matrix $x \in \mathbb{R}^{m \times n}$:

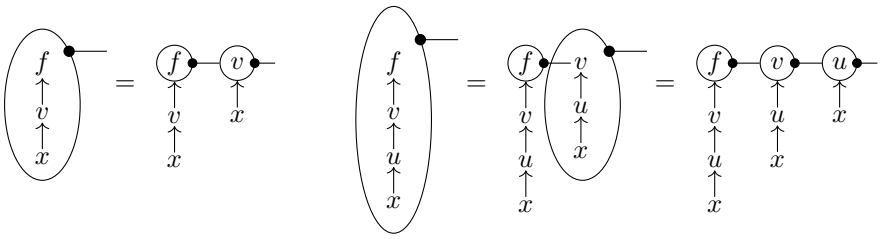
$$f \overset{m}{\underset{n}{\circlearrowleft}} x = \overbrace{f(x)}^{m} = \text{Tr}(f(x)).$$

4.1 The Chain Rule

One of the most attractive features of tensor diagrams is the transparency of the chain rule. For two functions $f : \mathbb{R}^m \to \mathbb{R}$ and $v : \mathbb{R}^n \to \mathbb{R}^m$, the composite function $f \circ v : \mathbb{R}^n \to \mathbb{R}$ is diagrammed simply by feeding the output of v into f:

$$f \leftarrow v \leftarrow x$$

In traditional notation, the chain rule is written $J_{f\circ v}(x) = \nabla f(v(x))\, J_v(x)$, where $J_v(x)$ is the Jacobian of v and $\nabla f(v(x))$ is the gradient of f at $v(x)$. With tensor diagrams *the chain rule is actually a chain!*



This clearly follows the classical mnemonic: "the derivative of the outer function times the derivative of the inner function". The only refinement we need is that the "outer function" is connected to the "inner function" using the new edge created by taking the derivative.

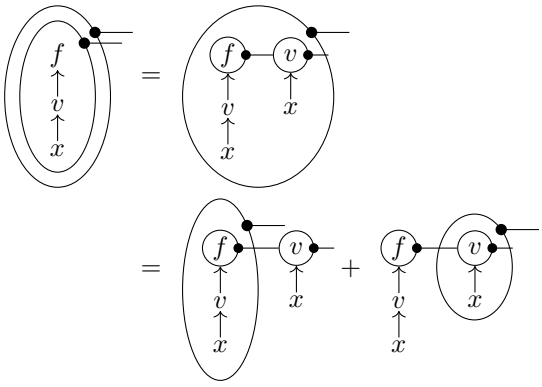
We could further simplify (f)— to f' —, where f' is the derivative of f. For example, if f was $\sin -$, then f' would be $\cos -$. But we can also just keep the circled f as an easy to recognize symbol for the derivative.

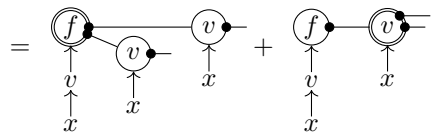
4.1.1 The Hessian Chain Rule

Things get even more interesting when we consider the second derivative. Using standard notation, the Hessian of $f \circ v$ is

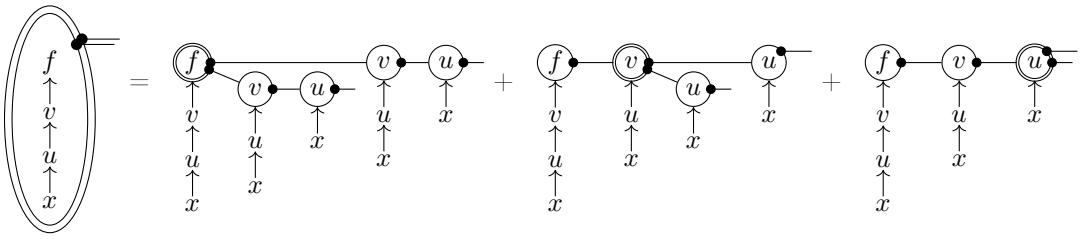
$$H_{f\circ v}(x) = Dv(x)^T \cdot D^2 f(v(x)) \cdot Dv(x) + \sum_{k=1}^{m} \frac{\partial f}{\partial v_k}(v(x)) \frac{\partial^2 v_k}{\partial x \partial x^T}(x).$$

This "Hessian Chain Rule" is much easier to derive using tensor diagrams:





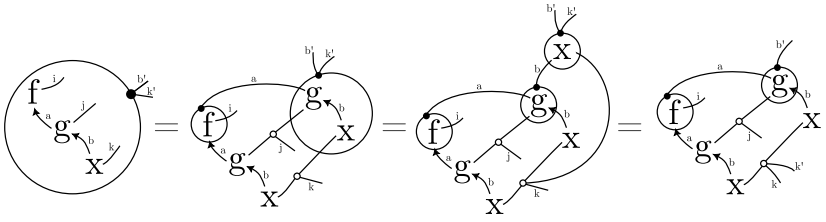
If we continue to expand this way, we see there is a simple pattern in the chain rule for more functions:



Yaroslav Bulatov has written extensively on the Hessian Chain Rule, and how to evaluate it efficiently for deep learning applications. Interested readers may refer to his blog post on the topic.

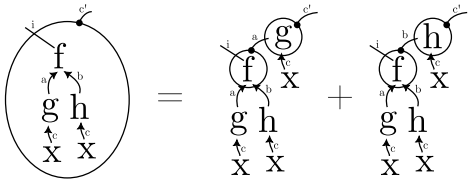
4.1.2 Chain Rule with Broadcasting

When the chain rule is applied to functions with broadcasting, the result still follows the "outer derivative times inner derivative" mnemonic, but the meaning of "times" is expanded to take the Hadamard product of the broadcasted edges.



4.1.3 Functions with multiple inputs

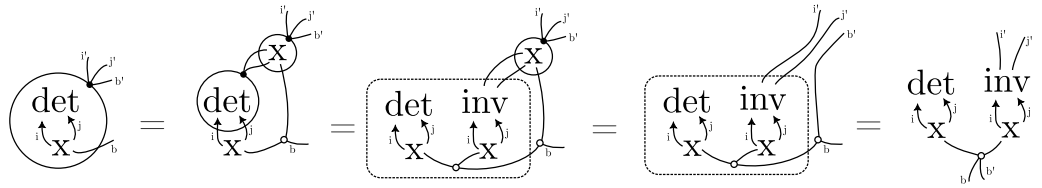
If $f : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ is a function with two inputs, the derivative of $f(x, y)$ is simply $\frac{\partial f}{\partial x}\partial x + \frac{\partial f}{\partial y}\partial y$, or in tensor diagram notation:

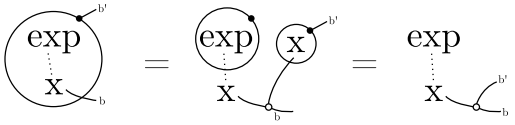


4.2 Examples

4.2.1 Known derivatives

Two examples where we can expand f with known derivatives:





Here $\mathrm{inv}(X)$ is the matrix inverse function, taking a matrix X to its inverse X^{-1}. In the case of the determinant, the first derivative is the classical

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

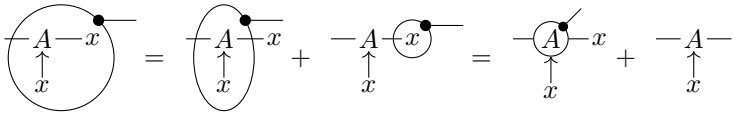
One may continue taking derivatives, but the pattern is not simply $\det(X)(X^{-1})^{\otimes k}$: each derivative introduces signs. For $k = 2$,

$$\frac{\partial^2 \det(X)}{\partial X_{ij}\,\partial X_{kl}} = \det(X)\left(X^{-1}_{ji}X^{-1}_{lk} - X^{-1}_{jk}X^{-1}_{li}\right),$$

and in general the k-th derivative is $\det(X)$ times the signed sum, over all permutations, of the possible wirings of k copies of X^{-1}.

4.2.2 Pseudo-linear forms

Pseudo-linear forms, $A(x)x$, are common. All pixel-adaptive filters like non-local means, bilateral, etc, and the so-called attention mechanism in transformers can be written this way. According to Peyman Milanfar, the gradient of this function is important and has a form worth remembering:



We may appreciate the simplicity of this expression, when we consider the following derivation given by Peyman Milanfar using classical notation:

$$\begin{aligned}\mathrm{d}[A(x)x] &= \mathrm{d}[A(x)]x + A(x)\mathrm{d}x \\ &= \mathrm{vec}(\mathrm{d}[A(x)]\,x) + A(x)\mathrm{d}x\end{aligned}$$

$$
\begin{aligned}
&= \mathrm{vec}(I\,\mathrm{d}[A(x)]\,x) + A(x)\mathrm{d}x \\
&= (x^T \otimes I)\,\mathrm{vec}(\mathrm{d}[A(x)]) + A(x)\mathrm{d}x \\
&= (x^T \otimes I)\mathrm{D}\,\mathrm{vec}[A(x)]\mathrm{d}x + A(x)\mathrm{d}x \\
&= [(x^T \otimes I)\mathrm{D}\,\mathrm{vec}[A(x)] + A(x)]\mathrm{d}x
\end{aligned}
$$

Which finally implies:

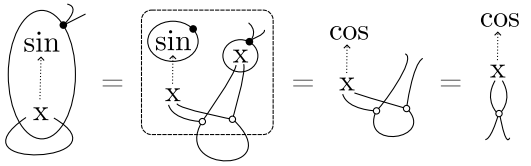
$$\frac{\partial A(x)x}{\partial x} = (x^T \otimes I)\frac{\partial}{\partial x}\mathrm{vec}[A(x)] + A(x).$$

4.2.3 Trace identity

The Matrix Cookbook has the formula:

$$\frac{\partial \mathrm{Tr}(\sin(X))}{\partial X} = \cos(X)^T \tag{128}$$

If we attempt to derive this using tensor diagrams, we get the following different result,



which is equivalent to $\cos(X) \circ I$. That is, $\cos(X)$ but zero everywhere except the diagonal.

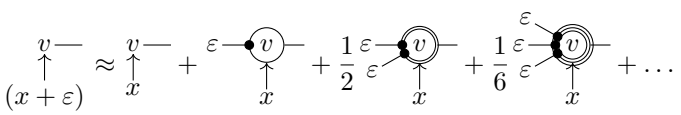
The reason is that the matrix cookbook actually uses a slightly different definition of "function applied to matrix". If F can be written as a power series, then one way to define $F(X)$ is the matrix power series:

$$F(X) = \sum_{k=0}^{\infty} a_k X^k.$$

In this case, the derivative of $\mathrm{Tr}(F(X))$ is $f(X)^T$, where $f(X)$ is the scalar derivative of $F(X)$, matching the Matrix Cookbook's formula.

4.2.4 Taylor

For a sufficiently differentiable function $v : \mathbb{R}^d \to \mathbb{R}^m$ we can write the Taylor expansion:



Writing this using classical notation is quite messy:

$$v(x+\varepsilon) \approx v(x) + \left[\frac{\partial}{\partial x}v(x)\right]\varepsilon + \frac{1}{2}\left[\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}v(x)\right]\varepsilon\right]\varepsilon + \frac{1}{6}\left[\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}v(x)\right]\varepsilon\right]\varepsilon\right]\varepsilon + \ldots$$

$$= v(x) + \left[\frac{\partial}{\partial x} v(x)\right] \varepsilon + \frac{1}{2}(\varepsilon^T \otimes I) \left[\frac{\partial \text{vec}}{\partial x} \left[\frac{\partial v(x)}{\partial x}\right]\right] \varepsilon + \frac{1}{6}(\varepsilon^T \otimes \varepsilon^T \otimes I) \left[\frac{\partial \text{vec}}{\partial x} \left[\frac{\partial \text{vec}}{\partial x} \left[\frac{\partial v(x)}{\partial x}\right]\right]\right] \varepsilon + \dots$$

With index notation it's ok:

$$v_i(x+\varepsilon) \approx v_i(x) + \sum_j \frac{\partial v_i(x)}{\partial x_j} \varepsilon_j + \frac{1}{2} \sum_{j,k} \frac{\partial^2 v_i(x)}{\partial x_j \partial x_k} \varepsilon_j \varepsilon_k + \frac{1}{6} \sum_{j,k,\ell} \frac{\partial^3 v_i(x)}{\partial x_j \partial x_k \partial x_\ell} \varepsilon_j \varepsilon_k \varepsilon_\ell$$

4.3 Exercises

Exercise 19. Draw the tensor diagram for a function $f : \mathbb{R}^n \to \mathbb{R}$ that applies an elementwise nonlinearity (for instance, exp) followed by a summation over the components. Verify that the diagram corresponds to the conventional formula for the softmax denominator.

Exercise 20. Represent the composition of two functions

$$f : \mathbb{R}^m \to \mathbb{R} \quad \text{and} \quad v : \mathbb{R}^n \to \mathbb{R}^m,$$

using tensor diagrams. Then, using the diagrammatic chain rule, derive the expression for the derivative of $f \circ v$ with respect to $x \in \mathbb{R}^n$.

Exercise 21. For a matrix function $A(x)$ that depends on a vector x, use tensor diagrams to illustrate the derivative

$$\frac{\partial}{\partial x}[A(x)x],$$

and explain how the product rule is implemented in the diagram.

Exercise 22. Represent the KL-divergence term for a Variational Autoencoder (VAE) as

$$KL(\mu, \sigma) = -\frac{1}{2} \sum_i \Big(1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2\Big),$$

with parameters given by

$$\mu = Wx + c \quad \text{and} \quad \log \sigma^2 = Wx + c.$$

Derive the gradient $\nabla_W KL$ with respect to the weight matrix W. Be sure to keep track of dimensions and account for elementwise operations.

Exercise 23. Let the softmax function $s : \mathbb{R}^n \to \mathbb{R}^n$ be defined by

$$s_i(z) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}, \quad i = 1, \dots, n.$$

Prove that the Jacobian matrix of s is given by

$$\frac{\partial s_i}{\partial z_j} = s_i\big(\delta_{ij} - s_j\big),$$

where δ_{ij} is the Kronecker delta.

Exercise 24. Suppose $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$ are independent samples from a multivariate normal distribution $N(\mu, \Sigma)$. Write the log-likelihood function and derive the gradients $\nabla_\mu \ell$ and $\nabla_\Sigma \ell$. Then, by setting these gradients to zero, show that the maximum likelihood estimators are

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x^{(i)} \quad \text{and} \quad \hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} (x^{(i)} - \hat{\mu})(x^{(i)} - \hat{\mu})^T.$$

Exercise 25. Consider a Gaussian process with covariance matrix $K(\theta)$ and the log-marginal likelihood defined as

$$L(\theta) = -\frac{1}{2}\Big(y^T K(\theta)^{-1} y + \log\det K(\theta)\Big).$$

Derive the gradient of L with respect to θ, showing that

$$\frac{\partial L}{\partial \theta} = \frac{1}{2}\operatorname{tr}\Big(\Big(K^{-1} y y^T K^{-1} - K^{-1}\Big)\frac{\partial K}{\partial \theta}\Big).$$

Exercise 26. In logistic regression, let

$$p_i = \sigma(w^T x_i) \quad \text{with} \quad \sigma(z) = \frac{1}{1+e^{-z}},$$

and consider the negative log-likelihood

$$J(w) = -\sum_{i=1}^{n} \Big[y_i \ln p_i + (1-y_i)\ln(1-p_i)\Big].$$

Derive the gradient $\nabla_w J$ and the Hessian $H = \nabla_w^2 J$. In particular, show that

$$\nabla_w J = \sum_{i=1}^{n} (p_i - y_i) x_i, \quad \text{and} \quad H = \sum_{i=1}^{n} p_i(1-p_i)\, x_i x_i^T.$$

Exercise 27. Let $X \in \mathbb{R}^{n\times n}$ be an invertible matrix. Prove that

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

In other words, show that the gradient of $\det(X)$ with respect to X is $\det(X)$ times the transpose of X^{-1}.

Exercise 28. For an invertible matrix $X \in \mathbb{R}^{n\times n}$, prove that

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

Briefly discuss why this result is useful in statistical applications such as Gaussian likelihoods.

Exercise 29. Let $A \in \mathbb{R}^{n\times n}$ be an invertible matrix (which may depend on a parameter). Starting from the identity $I = A^{-1}A$, differentiate both sides to show that

$$d(A^{-1}) = -A^{-1}\,(dA)\,A^{-1}.$$

Deduce an expression for the partial derivative $\frac{\partial A^{-1}}{\partial A_{ij}}$.

Exercise 30. Define the scalar function

$$f(A) = a^T A^{-1} b,$$

where $a, b \in \mathbb{R}^n$ are fixed vectors and $A \in \mathbb{R}^{n \times n}$ is invertible. Using the result from differentiating the inverse, show that

$$\nabla_A f(A) = -A^{-T}(a\, b^T)A^{-T}.$$

Exercise 31. Let $X \in \mathbb{R}^{n \times n}$ be a symmetric matrix with a simple eigenvalue λ and corresponding unit eigenvector v. Prove that the derivative of λ with respect to X is given by

$$\nabla_X \lambda = v\, v^T.$$

Additionally, discuss why this result does not hold when λ has multiplicity greater than one.

Exercise 32. For a symmetric matrix $A \in \mathbb{R}^{n \times n}$, consider the Rayleigh quotient

$$R(x) = \frac{x^T A x}{x^T x}, \quad x \in \mathbb{R}^n \setminus \{0\}.$$

Using Lagrange multipliers, show that the stationary values of $R(x)$ correspond to the eigenvalues of A, and that the maximizer (minimizer) is the eigenvector associated with the largest (smallest) eigenvalue.

Exercise 33. Let $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{n \times n}$ be constant matrices, and let $X \in \mathbb{R}^{m \times n}$ be a variable matrix. Prove that

$$\nabla_X \operatorname{Tr}(X^T A X B) = A X B + A^T X B^T.$$

(Hint: Use the cyclic property of the trace to rearrange terms.)

Exercise 34. (a) Show that for a constant matrix $A \in \mathbb{R}^{m \times n}$ and variable $X \in \mathbb{R}^{m \times n}$,

$$\frac{\partial}{\partial X} \operatorname{Tr}(A^T X) = A.$$

(b) More generally, for constant matrices $A \in \mathbb{R}^{p \times m}$ and $B \in \mathbb{R}^{n \times q}$, prove that

$$\nabla_X \operatorname{Tr}(A X B) = A^T B^T.$$

Exercise 35. Let $s \in \mathbb{R}^n$ be a vector and $C \in \mathbb{R}^{n \times n}$ be a fixed matrix. Define the matrix function

$$F(s) = \operatorname{diag}(s)\, C \operatorname{diag}(s),$$

where $\operatorname{diag}(s)$ denotes the diagonal matrix with the entries of s. Express the differential dF in terms of s, ds, and C, and hence derive an expression for the derivative $\nabla_s F(s)$.

Exercise 36. Let $A \in \mathbb{R}^{p \times m}$, $X \in \mathbb{R}^{m \times n}$, and $B \in \mathbb{R}^{q \times n}$. Prove the identity

$$\operatorname{vec}(A\, X\, B^T) = (B \otimes A)\, \operatorname{vec}(X).$$

Discuss how this identity can be used to convert certain matrix derivative problems into vectorized forms.

Exercise 37. Let

$$f(x) = x^T A x,$$

where $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ is not necessarily symmetric. Show that

$$\nabla_x f(x) = (A + A^T)x.$$

Then, compute the Hessian $\nabla_x^2 f(x)$ and discuss what simplification occurs when A is symmetric.

Exercise 38. Consider the Frobenius norm $\|X\|_F = \sqrt{\mathrm{Tr}(X^T X)}$, for $X \in \mathbb{R}^{m \times n}$.

Show that

$$\nabla_X \|X\|_F = \frac{X}{\|X\|_F}$$

for $X \neq 0$.

Exercise 39. Let $Y \in \mathbb{R}^{n \times n}$ be a symmetric matrix, and define $\phi(Y)$ to be its largest eigenvalue. If this eigenvalue has multiplicity $k > 1$, the function ϕ is not differentiable at Y. Show that for any unit eigenvector u corresponding to the largest eigenvalue, the matrix

$$G = u\, u^T$$

is a valid subgradient of ϕ at Y. More generally, show that any convex combination $\sum_{i=1}^k c_i\, u_i u_i^T$, where $\{u_i\}_{i=1}^k$ is an orthonormal basis for the eigenspace corresponding to the largest eigenvalue, $c_i \geq 0$ and $\sum_{i=1}^k c_i = 1$, is a valid subgradient. Explain the challenges that arise in defining a unique gradient in this setting.

Chapter 5

Statistics and Probability

5.1 Definition of Moments

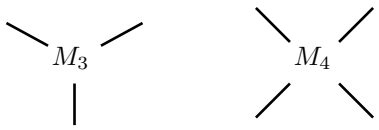
Let $x \in \mathbb{R}^n$ be a random variable. We write $m = E[x] \in \mathbb{R}^n$ for the expectation and write $\tilde{x} = x - m$ for the centered variable, so that $M = \mathrm{Var}[x] = E[\tilde{x}\tilde{x}^T]$ is the covariance (when these quantities are defined.)

In tensor diagrams, we will use square brackets:

$$—m = [—x] \quad \text{and} \quad —M— = [—\tilde{x} \;\; \tilde{x}—]$$

Inside diagrams we always write the centered variable as $\tilde{x}$ rather than $x - m$, since a bare minus sign inside a diagram could be mistaken for a contraction edge.

We can also define the third and fourth centralized moment tensors



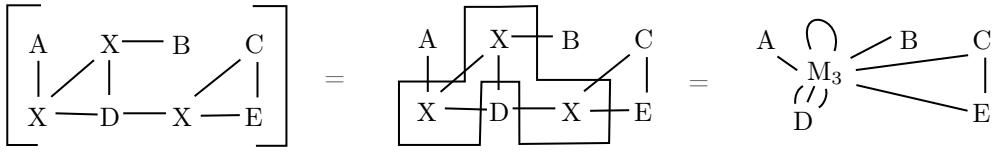
equal to the natural outer products

$$M_3 = \begin{bmatrix} \tilde{x}— \\ \tilde{x}— \\ \tilde{x}— \end{bmatrix} \quad \text{and} \quad M_4 = \begin{bmatrix} \tilde{x}— \\ \tilde{x}— \\ \tilde{x}— \\ \tilde{x}— \end{bmatrix}.$$

5.1.1 Expectation of Linear Combinations

General principle: The "linearity of expectation" lets you pull out all parts of the graph not involving X, which we assume here has mean zero, so that its third moment tensor

is exactly the M_3 defined above.



The remaining tensor on the right (labeled M_3 in the figure) is an order-9 tensor that depends only on the distribution of X, not on the constants A, B, C, D, E. In practice you would want to name the edges to keep track of what gets multiplied with what.

We can even use linearity of expectation to push the expectation inside an infinite sum of tensors, as in the following moment generating function, which relates all the M_k tensors:

$$\mathrm{E}(\mathrm{e}^{\langle \tilde{x}, t\rangle}) = \sum_{k=0}^{\infty} \frac{1}{k!} \mathrm{E}[\langle \tilde{x}, t\rangle^k] = \mathrm{E}\big[\sum_{k=0}^{\infty} \frac{1}{k!} \langle \tilde{x}^{\otimes k}, t^{\otimes k}\rangle\big] = \sum_{k=0}^{\infty} \frac{1}{k!} \langle M_k, t^{\otimes k}\rangle$$

$$= 1 + \frac{1}{2}\, t - M - t + \frac{1}{6}\, t - M_3 \,\diagdown t \,\diagdown t + \frac{1}{24}\, t \,\diagdown M_4 \,\diagdown t \,\diagdown t + \dots$$

Note that there is no linear term, since $M_1 = \mathrm{E}[\tilde{x}] = 0$.

5.1.2 Linear Forms

The Matrix Cookbook gives the following simple expectation:

$$\mathrm{E}[AXB + C] = A\mathrm{E}[X]B + C \qquad \begin{bmatrix} -A-X-B- \\ +\ -C- \end{bmatrix} = \begin{matrix} -A-[X]-B- \\ +\ -C- \end{matrix} \tag{312}$$

5.1.3 Quadratic Forms

We often prefer to write expectations in terms of the simple centered moments, which we can do by pulling out the mean:

$$\begin{bmatrix} x- \\ x- \end{bmatrix} = \begin{bmatrix} \tilde{x}- \\ \tilde{x}- \end{bmatrix} + \begin{matrix} m- \\ m- \end{matrix} = M \diagup\!\diagdown + \begin{matrix} m- \\ m- \end{matrix}$$

This makes it easy to handle the quadratic forms from the Matrix Cookbook:

$$\mathrm{Var}[Ax] = A\mathrm{Var}[x]A^T \qquad \begin{bmatrix} -\widetilde{Ax} \\ -\widetilde{Ax} \end{bmatrix} = \begin{bmatrix} A-\tilde{x} \\ A-\tilde{x} \end{bmatrix} \tag{313}$$

$$= \begin{bmatrix} A—\tilde{x} \\ A—\tilde{x} \end{bmatrix}$$

$$= -A-M_2-A-$$

$$\mathrm{E}[x^T A x] = \mathrm{Tr}(AM) + m^T A m \qquad [x\text{–}A\text{–}x] = \left(\begin{bmatrix}\tilde{x}\text{–}\\ \tilde{x}\text{–}\end{bmatrix} + \begin{matrix} m\text{–}\\ m\text{–}\end{matrix}\right) \!>\! A \tag{318}$$

$$= \begin{bmatrix} \tilde{x} \\ \tilde{x} \end{bmatrix} \!\!\diagdown\!\!\diagup\!\! A + \begin{matrix} m \\ m \end{matrix} \!\!\diagdown\!\! A$$

$$= \overbrace{M \text{—} A} \; + m\text{–}A\text{–}m$$

The general outer-product form is obtained by the same "pull out the mean" step applied to two independent linear expressions. We split $Ax + a = A\tilde{x} + (Am + a)$ and likewise for $Bx + b$; the centered-times-mean cross terms vanish in expectation.

$$\mathrm{E}[(Ax + a)(Bx + b)^T] \qquad \begin{bmatrix} A\text{–}x + a \\ B\text{–}x + b \end{bmatrix} \tag{320}$$

$$= AMB^T + (Am + a)(Bm + b)^T \qquad = \begin{bmatrix} A\text{–}\tilde{x} \\ B\text{–}\tilde{x} \end{bmatrix} + \begin{bmatrix} A\text{–}m + a \\ B\text{–}m + b \end{bmatrix}$$

$$= \text{–}A\text{–}M_2\text{–}B\text{–} + \begin{bmatrix} A\text{–}m + a \\ B\text{–}m + b \end{bmatrix}$$

All the other outer-product identities are specializations of (320) with A, B set to I or to each other, and a, b optionally zero.

$$\mathrm{E}[xx^T] = M + mm^T \qquad \begin{bmatrix} x\text{–} \\ x\text{–} \end{bmatrix} = M \!\!\diagdown\!\!\diagup\!\! + \begin{matrix} m\text{–} \\ m\text{–} \end{matrix} \tag{321}$$

$$\mathrm{E}[(Ax)(Ax)^T] = A(M + mm^T)A^T \qquad \begin{bmatrix} A\text{–}x \\ A\text{–}x \end{bmatrix} = \text{–}A\text{–}M_2\text{–}A\text{–} + \begin{bmatrix} A\text{–}m \\ A\text{–}m \end{bmatrix} \tag{324}$$

$$\mathrm{E}[(x + a)(x + a)^T] = M + (m + a)(m + a)^T \qquad \begin{bmatrix} x + a \\ x + a \end{bmatrix} = M \!\!\diagdown\!\!\diagup\!\! + \begin{matrix} m + a \\ m + a \end{matrix} \tag{325}$$

The remaining two outer-product identities are just (321) contracted against a vector a (so one of the two free edges gets closed):

$$\mathrm{E}[x\, a^T x] = (M + mm^T)a \qquad \text{–}\mathrm{E}[xx^T]\text{–}a \;=\; \text{–}M_2\text{–}a \;+\; \text{–}m\text{–}m\text{–}a \tag{322}$$

$$\mathrm{E}[x^T a\, x^T] = a^T(M + mm^T) \qquad \text{(transpose of (322))} \tag{323}$$

Contracting the two free edges of (320) against each other (i.e., taking the trace) turns the outer-product identities into scalar inner-product identities. The inner product $(Ax + a)^T(Bx + b)$ is just the two vectors joined on their free edges:

$$\mathrm{E}[(Ax+a)^T(Bx+b)] \qquad (Ax+a)\text{–}(Bx+b) \tag{326}$$

$$= \mathrm{Tr}(AMB^T) + (Am+a)^T(Bm+b) \qquad = \overbrace{A\text{—}M_2\text{—}B} + (Am+a)\text{–}(Bm+b)$$

The remaining trace identities specialize (326) the same way:

$$\mathrm{E}[x^Tx] \qquad x\text{–}x \tag{327}$$

$$= \mathrm{Tr}(M) + m^Tm \qquad = \overbrace{\quad M\quad} + \; m\text{–}m$$

$$\mathrm{E}[x^TAx] = \mathrm{Tr}(AM) + m^TAm \qquad \text{(already shown as (318))} \tag{328}$$

$$\mathrm{E}[(Ax)^T(Ax)] \qquad (Ax)\text{–}(Ax) \tag{329}$$

$$= \mathrm{Tr}(AMA^T) + (Am)^T(Am) \qquad = \overbrace{A\text{—}M_2\text{—}A} + \; (Am)\text{–}(Am)$$

$$\mathrm{E}[(x+a)^T(x+a)] \qquad (x+a)\text{–}(x+a) \tag{330}$$

$$= \mathrm{Tr}(M) + (m+a)^T(m+a) \qquad = \overbrace{\quad M\quad} + \; (m+a)\text{–}(m+a)$$

5.1.4 Cubic Forms

The raw third moment $\mathrm{E}[x^{\otimes 3}]$ decomposes into central moments by the same "pull out the mean" expansion, $x = \tilde{x} + m$. Of the eight terms in the expansion of $(\tilde{x}+m)^{\otimes 3}$, the three terms containing exactly one centered factor vanish in expectation (since $\mathrm{E}[\tilde{x}] = 0$), leaving

$$\begin{bmatrix} x- \\ x- \\ x- \end{bmatrix} = \begin{bmatrix} \tilde{x}- \\ \tilde{x}- \\ \tilde{x}- \end{bmatrix} + 3 \cdot \underbrace{\begin{pmatrix} \begin{bmatrix} \tilde{x}- \\ \tilde{x}- \end{bmatrix} \\ m- \end{pmatrix}}_{\text{one of three positions}} + \begin{matrix} m- \\ m- \\ m- \end{matrix}$$

$$= \overset{|}{M_3} \;+\; 3 \;\; \begin{matrix} m \\ \\ M \end{matrix} \;\; + \;\; \begin{matrix} m- \\ m- \\ m- \end{matrix}$$

The squiggle denotes symmetrization over its edges (the average over all $3! = 6$ permutations of those legs, following the convention in §1.7). Since M is symmetric, permutations that only swap M's two legs act trivially, so the average produces each of the 3 distinct assignments of m and M to the legs with weight $\frac{2}{6} = \frac{1}{3}$; the factor 3 turns this average back into the sum of the three distinct terms.

For an affine expression $(Ax+a)(Bx+b)^T(Cx+c)$, the same pattern gives

$$\mathrm{E}\big[(Ax+a)(Bx+b)^T(Cx+c)\big] = A \cdot M_3\big(B^T, C\big)$$

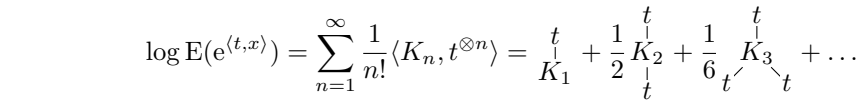
$$+ AMB^T(Cm+c) + AMC^T(Bm+b) + (Am+a)\operatorname{Tr}(BMC^T)$$
$$+ (Am+a)(Bm+b)^T(Cm+c),$$

where $M_3(B^T, C)$ denotes the order-1 tensor obtained by contracting two legs of M_3 with B^T and C respectively, and then contracting the remaining free indices of B and C with each other; in components, $M_3(B^T, C)_i = (M_3)_{ijk} B_{lj} C_{lk}$.

5.2 Cumulants

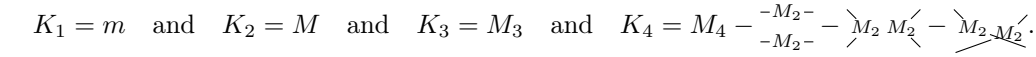
Given a random vector $x \in \mathbb{R}^d$, its nth Cumulant Tensor, $K_n \in \mathbb{R}^{d,\dots,d}$ is defined by

$$\log \mathrm{E}(\mathrm{e}^{\langle t,x\rangle}) = \sum_{n=1}^{\infty} \frac{1}{n!}\langle K_n, t^{\otimes n}\rangle =$$



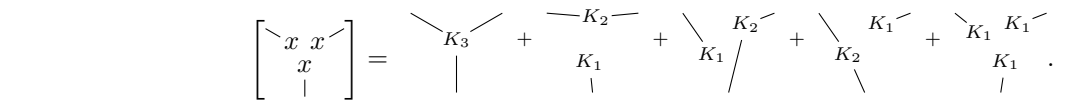
The first couple of cumulants are similar to the central moments:

$K_1 = m$ and $K_2 = M$ and $K_3 = M_3$ and $K_4 = M_4 -$



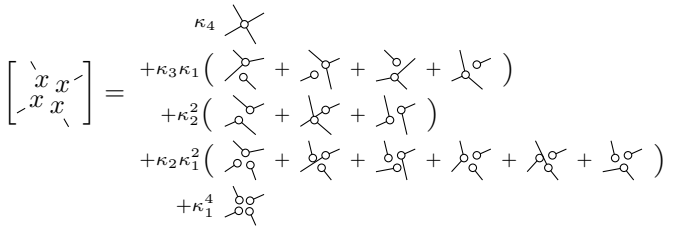
They have the nice property (which is easy to see from the definition) that they are additive for independent random variables: $K_n(x+y) = K_n(x) + K_n(y)$. This generalizes the standard property that the variance of the sum of independent random variables is the sum of the variances.

We can write the expectations of $x^{\otimes n}$ in terms of the cumulants:

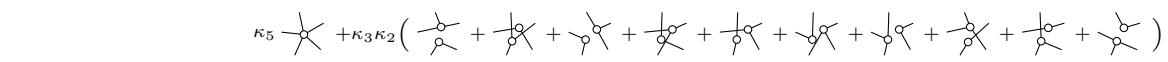


In general the sum is over all the partitions of the set $\{1, \dots, n\}$.

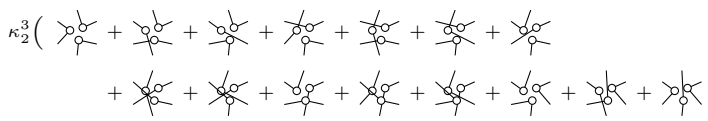
If the entries of x are *independent* and identically distributed with scalar cumulants $\kappa_1, \kappa_2, \kappa_3, \kappa_4 \in \mathbb{R}$, the off-diagonals of the cumulant tensors $K_1, K_2, \dots$ all vanish, and the partition sum reduces to a scalar weight times a "copy tensor" (diagonal) for each block. For $n = 4$:



Note in particular, that if the mean, κ_1, is zero, only four terms survive. For $n = 5$ there are 52 partitions in total, but only 11 survive if $\kappa_1 = 0$:



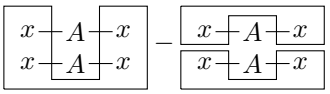
If x is a centered (zero-mean) Gaussian, all cumulants except κ_2 are zero. For $n = 6$ there are 203 partitions in total, but only 15 terms[1] for $E[x^{\otimes 6}]$:



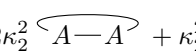
5.2.1 Quartic Forms

We can use this to compute $\mathrm{Var}[x^T Ax]$. We assume A is symmetric and, as above, that the entries of x are independent and identically distributed with scalar cumulants $\kappa_1, \kappa_2, \kappa_3, \kappa_4$:

$$\mathrm{Var}[x^T Ax] =$$

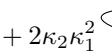


$$= \kappa_4 A \bigcirc\!\!\!\!\bigcirc A + 4\kappa_3\kappa_1 A \bigcirc\!\!\!\!-\, A \multimap$$



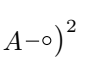
$$+ \kappa_2^2$$

$$+ 4\kappa_2\kappa_1^2 \circ\!\!- A - A \multimap + 2\kappa_2\kappa_1^2$$



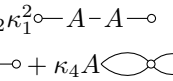
$$\circ - A - \circ + \kappa_1^4 \circ - A - \circ \circ - A - \circ$$

$$- \Big(\kappa_2$$



$$+ \kappa_1^2 \circ - A - \circ\Big)^2$$

$$= 2\kappa_2^2$$



$$+ 4\kappa_2\kappa_1^2 \circ\!\!- A - A \multimap$$

$$+ 4\kappa_3\kappa_1 A \bigcirc\!\!\!\!-\, A \multimap + \kappa_4 A \bigcirc\!\!\!\!\bigcirc A$$

The Matrix Cookbook lists this as

$$\mathrm{Var}[x^T Ax] = 2\mu_2^2 \mathrm{Tr}(A^2) + 4\mu_2 c^T A^2 c + 4\mu_3 c^T Aa + (\mu_4 - 3\mu_2^2) a^T a \tag{319}$$

where $c = \mu_1 \mathbf{1}$ is the mean vector of x (every coordinate has mean μ_1), and $a = \mathrm{diag}(A)$ is the diagonal of A, and $\mu_4 = \kappa_4 + 3\kappa_2^2$ is the fourth central moment.

5.3 Weighted Scalar Variable

If we contract the random vector x with a deterministic vector w to form a scalar $y = w^T x$, the central moments of y are just the central-moment tensors of x contracted on all legs with w:

$$\begin{aligned} \mathrm{E}[y] &= w^T m \\ \mathrm{E}[(y - m_y)^2] &= w - M_2 - w \\ \mathrm{E}[(y - m_y)^3] &= \overset{w}{\underset{|}{w}} - M_3 - w \end{aligned}$$

[1] This is why $\mathrm{E}[g^6] = 15$ for $g \sim N(0, 1)$.

$$\mathrm{E}[(y-m_y)^4] = \overset{w}{\underset{w}{\overset{|}{\underset{|}{}}}} w\text{-}M_4\text{-}w$$

where $m_y = \mathrm{E}[y] = w^T m$.
For $x \sim N(0, I)$ (unit Gaussian) all central moments M_k with k odd vanish, and $\mathrm{E}[|y|^n]^{1/n} \le \sqrt{2n/\pi}\,\|w\|_2$ by the Gaussian moment formula $\mathrm{E}|g|^n = \sigma^n 2^{n/2}\Gamma(\frac{n+1}{2})/\sqrt{\pi}$.

5.4 Gaussian Moments

For a Gaussian vector $x \sim \mathcal{N}(m, M)$, all cumulants beyond the second vanish: $K_1 = m$, $K_2 = M$, and $K_n = 0$ for $n \ge 3$. The moment-cumulant formula from the Cumulants section then collapses to a sum over partitions of $\{1, \dots, n\}$ into singletons (each carrying a factor of m) and pairs (each carrying a factor of M) — *Isserlis' theorem*, also known as *Wick's theorem*:

$$\left[\overset{n \text{ copies}}{\overset{|}{x}\ \cdots\ \overset{|}{x}}\right] = \sum_{\pi \in \mathcal{P}(n)} \prod_{B \in \pi} \begin{cases} m & |B| = 1 \\ M & |B| = 2 \\ 0 & |B| \ge 3 \end{cases}.$$

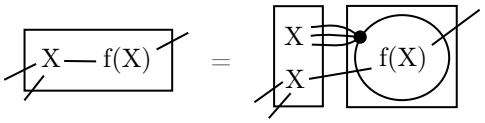
For centered Gaussians ($m = 0$) only the pairings survive, giving the familiar identity

$$E[x^{\otimes 2k}] = \sum_{\pi \text{ pairing}} \bigotimes_{(i,j)\in\pi} M,$$

a sum with $(2k-1)!!$ terms (3 for $k = 2$, 15 for $k = 3$).

5.4.1 Gaussian Integration by Parts

If X is a tensor with Gaussian entries, zero mean, and some covariance, Stein's lemma gives the following very general equation, for any differentiable function f:



Combined with the tensor chain rule from chapter 4, this can be a very powerful way to evaluate many hard expectations.

5.4.2 Quadratic and Quartic Forms

The quadratic-form identities of the Quadratic Forms section hold for arbitrary distributions, so they apply to Gaussians without modification — see (318) and (321). What *is* Gaussian-specific is the variance of a quadratic form, where, for symmetric A, Wick pairing leaves only two surviving terms after cancelling $\mathrm{E}[x^T Ax]^2$:

$$\mathrm{Var}[x^T Ax] = 2\,\mathrm{Tr}(AMAM) + 4\,m^T AMAm \tag{379}$$

$$= 2 \quad \overset{\frown}{A—M—A—M} \quad + \ 4\ m–A–M–A–m\,.$$

Specializing to $M = \sigma^2 I$ recovers the Matrix Cookbook identity

$$\mathrm{Var}[x^T A x] = 2\sigma^4 \mathrm{Tr}(A^2) + 4\sigma^2 m^T A^2 m. \tag{381}$$

For two symmetric forms with the same Gaussian,

$$\mathrm{Cov}(x^T A x,\, x^T B x) = 2\,\mathrm{Tr}(AMBM) + 4\, m^T AMBm. \tag{382}$$

(Our version generalizes the Matrix Cookbook's (382), which assumes $x \sim \mathcal{N}(0, \sigma^2 I)$ and hence reads $2\sigma^4 \mathrm{Tr}(AB)$.)

More general fourth moments follow the same Wick-pairing recipe: given any four linear expressions $\ell_1(x), \ldots, \ell_4(x)$, we have

$$\mathrm{E}[\ell_1 \ell_2 \ell_3 \ell_4] = \sum_{3 \text{ pairings}} \mathrm{E}[\ell_i \ell_j]\, \mathrm{E}[\ell_k \ell_l] + \text{mean-only terms}.$$

For example,

$$\begin{aligned}\mathrm{E}[a^T x\, b^T x\, c^T x\, d^T x] = {} & (a^T M b)(c^T M d) + (a^T M c)(b^T M d) + (a^T M d)(b^T M c) \\ & + \text{mean-dependent terms},\end{aligned}$$

which the reader can unfold by writing $\ell_i = a_i^T \tilde{x} + a_i^T m$ and collecting terms.

5.4.3 Mixture of Gaussians

For a mixture $x \sim \sum_k \rho_k \mathcal{N}(m_k, M_k)$, linearity of expectation gives

$$\mathrm{E}[x] = \sum_k \rho_k\, m_k \tag{384}$$

$$\mathrm{Var}[x] = \sum_k \rho_k (M_k + m_k m_k^T) - \left(\sum_k \rho_k m_k\right)\left(\sum_k \rho_k m_k\right)^T. \tag{385}$$

Higher moments follow the same pattern — each component contributes its own Wick-pairing sum, weighted by ρ_k.

5.4.4 Derivatives of the Gaussian Log-Likelihood

For $x \sim \mathcal{N}(m, M)$ with density

$$p(x;\, m, M) = (2\pi)^{-n/2} \det(M)^{-1/2} \exp\!\big(-\tfrac{1}{2}(x-m)^T M^{-1}(x-m)\big)\,,$$

the log-likelihood score functions are

$$\begin{aligned}\frac{\partial \log p}{\partial m} &= M^{-1}(x-m), \\ \frac{\partial \log p}{\partial M} &= \tfrac{1}{2}\, M^{-1}\big[(x-m)(x-m)^T - M\big] M^{-1}.\end{aligned}$$

Both have zero expectation under p, as scores must. Setting the sample scores to zero, the MLE estimators from N i.i.d. samples $x_1, \dots, x_N$ are

$$\hat{m} = \frac{1}{N}\sum_i x_i, \qquad \hat{M} = \frac{1}{N}\sum_i (x_i - \hat{m})(x_i - \hat{m})^T.$$

The expected Fisher information follows from a second differentiation and Stein's lemma; it equals the covariance of the score, $\mathrm{E}\big[\nabla \log p\,(\nabla \log p)^T\big]$, which we can evaluate with the tensor Wick theorem above.

5.5 Free Probability for Random Matrices

When we move from scalar random variables to matrices, two things change: matrix products don't commute, and the dimension N becomes a parameter in its own right. The payoff: as $N \to \infty$, many matrix ensembles have limiting spectra that are governed by a clean analogue of the cumulants of §5.2 – *free cumulants* – where the sum-over-partitions gets restricted to *non-crossing* partitions. This single adjustment captures both the Wigner semicircle (free Gaussian) and the Marchenko–Pastur (free χ^2) distributions within the same formal framework.

5.5.1 Non-Crossing Partitions

Place n dots on a line. A partition π of $\{1, \dots, n\}$ is *non-crossing* if its blocks can be drawn as arcs above the dots without any two arcs intersecting. For $n = 4$, fourteen of the fifteen set partitions are non-crossing; the lone crossing partition is $\{1, 3\}, \{2, 4\}$:


(three pairings; only the middle one crosses.)

Write $\mathrm{NC}(n)$ for the set of non-crossing partitions of $\{1, \dots, n\}$. Its size is the n-th Catalan number $C_n = \binom{2n}{n}/(n+1) = 1, 1, 2, 5, 14, 42, \dots$

5.5.2 Free Moments and Free Cumulants

For an $N \times N$ random matrix X with orthogonal-conjugation-invariant distribution, define the *free moments* and *free cumulants* by

$$m_n(X) = \lim_{N\to\infty} \frac{1}{N}\,\mathrm{E}[\mathrm{Tr}(X^n)], \qquad m_n(X) = \sum_{\pi \in \mathrm{NC}(n)} \prod_{B \in \pi} \kappa^{\mathrm{free}}_{|B|}(X).$$

The second equation is Speicher's moment-cumulant formula – the same sum-over-partitions as in §5.2, but restricted to non-crossing ones. Writing κ_k for κ^{free}_k and expanding for small n:

$$\begin{aligned} m_1 &= \kappa_1, \\ m_2 &= \kappa_2 + \kappa_1^2, \\ m_3 &= \kappa_3 + 3\kappa_2\kappa_1 + \kappa_1^3, \\ m_4 &= \kappa_4 + 4\kappa_3\kappa_1 + 2\kappa_2^2 + 6\kappa_2\kappa_1^2 + \kappa_1^4. \end{aligned}$$

The coefficient 2 on κ_2^2 (not 3 as in the classical case) is the key difference: the pair-pair pattern has three set-partitions $\{1,2\}\{3,4\}, \{1,3\}\{2,4\}, \{1,4\}\{2,3\}$, but the middle one crosses and drops out.

5.5.3 R-Transform and Free Convolution

The *R-transform* of X is the generating function of free cumulants,

$$R_X(z) = \sum_{n \geq 1} \kappa_n^{\text{free}}(X)\, z^{n-1}.$$

The crucial property: if X and Y are *freely independent* (the non-commutative analogue of "classical independence" for large random matrices – essentially, their eigenvector bases are asymptotically generic with respect to each other), then

$$R_{X+Y}(z) = R_X(z) + R_Y(z).$$

This is the direct free analogue of the classical statement that cumulants add under independence. For free multiplicative convolution (the distribution of $X^{1/2} Y X^{1/2}$ when X, Y are free positive), the corresponding transform is the S-transform and is multiplicative.

5.5.4 Two Canonical Distributions

Wigner semicircle (free analogue of the standard Gaussian). Density $\rho(x) = \frac{1}{2\pi}\sqrt{4 - x^2}$ on $[-2, 2]$. All free cumulants vanish except $\kappa_2 = 1$, so $R(z) = z$. It arises as the $N \to \infty$ limit of the spectrum of an $N \times N$ Gaussian symmetric matrix (GOE/GUE) with entries of variance $1/N$. Just as the classical central limit theorem makes sums of i.i.d. scalars Gaussian, the *free* CLT makes sums of free self-adjoint matrices semicircular.

Marchenko–Pastur (free analogue of the scaled χ^2). For the Wishart matrix $W = \frac{1}{N} X X^T$ with $X \in \mathbb{R}^{N \times cN}$ having i.i.d. zero-mean unit-variance entries, the limiting spectrum is MP(c) with R-transform $R(z) = \frac{c}{1 - z}$.

5.5.5 A Note on Higher-Rank Tensors

For random tensors of rank ≥ 3 a partial analogue of free probability exists; see e.g. Gurau's *Random Tensors* (Oxford, 2017). The relevant combinatorial object is *colored non-crossing partitions*, counted by the Fuss–Catalan numbers $C_n^{(D)} = \frac{1}{(D-1)n+1}\binom{Dn}{n}$ (so $C_n^{(2)} = C_n$). The analogue of the non-crossing-pairing dominance for matrices is the dominance of *melonic* diagrams in the large-N limit. The theory is genuinely richer than the matrix case and we do not develop it further here.

5.6 Exercises

Exercise 40. Assume the coordinates of x are independent, and let v_3 denote the vector of their third central moments, $(v_3)_i = \mathrm{E}[(x_i - m_i)^3]$. Show that

$$\mathrm{E}[(Ax + a)(Ax + a)^T (Ax + a)] = A\, \mathrm{diag}(A^T A)\, v_3$$

$$+[2AMA^T + (Am+a)(Am+a)^T](Am+a)$$
$$+\mathrm{Tr}(AMA^T)(Am+a)$$

Assuming instead that x is Gaussian, show that

$$\begin{aligned}\mathrm{E}[(Ax+a)b^T(Cx+c)(Dx+d)^T] = (Am+a)b^T(CMD^T + (Cm+c)(Dm+d)^T)\\ +(AMC^T + (Am+a)(Cm+c)^T)b(Dm+d)^T\\ +b^T(Cm+c)(AMD^T - (Am+a)(Dm+d)^T)\end{aligned}$$

Exercise 41. Find more identities in the Matrix Reference Manual and try to prove them. Also try to verify your derivations using tensorgrad.

Exercise 42. Enumerate the non-crossing partitions of $\{1,2,3,4,5\}$ and verify there are $C_5 = 42$ of them. How many of the 52 set partitions of $\{1,\dots,5\}$ are crossing?

Exercise 43. Compute the free cumulants κ_n^{free} of the semicircle distribution by inverting the moment-cumulant formula on the Catalan moments $m_{2k} = C_k$, $m_{2k+1} = 0$.

Exercise 44. Show that if X has R-transform $R_X(z)$, then for a scalar $\lambda > 0$, λX has R-transform $R_{\lambda X}(z) = \lambda\, R_X(\lambda z)$.

Chapter 6

Determinant and Inverses

6.1 Determinant

It's convenient to write the determinant in tensor notation as

$$\det(A) = \overline{\begin{array}{ccc} \vert & & \vert \\ A & \cdots & A \\ \vert & & \vert \end{array}}$$

where the thick bars are the anti-symmetrizers of §1.7. Recall that, following Penrose's convention, the anti-symmetrizer is *normalized*: across n edges it denotes the idempotent projection $P = \frac{1}{n!}\sum_\sigma \operatorname{sign}(\sigma)\,\sigma$, so the closed diagram above computes $\operatorname{Tr}(PA^{\otimes n}P)$. When the thick bar appears on its own, as a tensor with n legs, it equals the Levi-Civita tensor scaled by $1/\sqrt{n!}$: $\underline{\begin{array}{cccc} \vert & \vert & & \vert \\ i_1 & i_2 & \cdots & i_n \end{array}} = \frac{1}{\sqrt{n!}}\,\varepsilon_{i_1,\ldots,i_n}$, where the raw rank-$n$ Levi-Civita tensor is defined by

$$\varepsilon_{i_1,\ldots,i_n} = \begin{cases} \operatorname{sign}(\sigma) & \sigma = (i_1,\ldots,i_n) \text{ is a permutation} \\ 0 & \text{otherwise,} \end{cases}$$

so that $P = \frac{1}{n!}\,\varepsilon \otimes \varepsilon$ when the dimension of the edges is $d = n$. The square root, which follows Cvitanović's birdtrack normalization [6], is what makes the two readings agree: stacking two bar-tensors gives $\frac{\varepsilon}{\sqrt{n!}} \otimes \frac{\varepsilon}{\sqrt{n!}} = P$ exactly, so we may evaluate any closed diagram either bar-by-bar or through the projector P and get the same value. To see that the definition makes sense, let's first consider

$$\det(I) = \overline{\begin{array}{ccc} \vert & & \vert \\ \vert & \cdots & \vert \end{array}} = \operatorname{Tr}(P) = \frac{1}{n!}\sum_{i_1,\ldots,i_n,j_1,\ldots,j_n} \varepsilon_{i_1,\ldots,i_n}\varepsilon_{j_1,\ldots,j_n}[i=j] = \frac{1}{n!}\sum_{i_1,\ldots,i_n} \varepsilon^2_{i_1,\ldots,i_n} = 1.$$

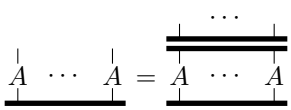
Indeed, P projects onto the space of anti-symmetric order-n tensors, which for $d = n$ is one-dimensional (spanned by ε), so its trace is 1. In general, since P is idempotent we have $\operatorname{Tr}(PA^{\otimes n}P) = \operatorname{Tr}(PA^{\otimes n})$, and expanding $P = \frac{1}{n!}\,\varepsilon \otimes \varepsilon$ recovers the permutation definition of the determinant:

$$\overline{\begin{array}{ccc} \vert & & \vert \\ A & \cdots & A \\ \vert & & \vert \end{array}} = \frac{1}{n!}\sum_{i_1,\ldots,i_n,j_1,\ldots,j_n} \varepsilon_{i_1,\ldots,i_n}\varepsilon_{j_1,\ldots,j_n} A_{i_1,j_1}\cdots A_{i_n,j_n}$$

$$
\begin{aligned}
&= \frac{1}{n!} \sum_{\sigma,\tau} \mathrm{sign}(\sigma)\mathrm{sign}(\tau) A_{\sigma_1,\tau_1} \cdots A_{\sigma_n,\tau_n} \\
&= \frac{1}{n!} \sum_{\sigma} \mathrm{sign}(\sigma) \sum_{\tau} \mathrm{sign}(\tau) A_{\sigma_1,\tau_1} \cdots A_{\sigma_n,\tau_n} \\
&= \frac{1}{n!} \sum_{\sigma} \mathrm{sign}(\sigma)^2 \det(A) \\
&= \det(A).
\end{aligned}
$$

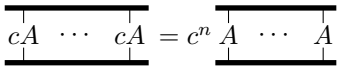
The definition generalizes to Cayley's "hyper determinants" by ....

A curious property is that

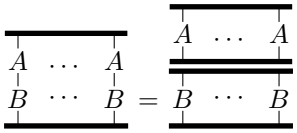


that is, applying $A^{\otimes n}$ to the bar-tensor just scales it by $\det(A)$. This is because the bar-tensor spans the one-dimensional image of P, a line which $A^{\otimes n}$ maps to itself.

(18) $\det(A) = \prod_i \lambda_i$ ...

(19) $\det(cA) = c^n \det(A)$ 

(20) $\det(A) = \det(A^T)$ ...

(21) $\det(AB) = \det(A)\det(B)$ 

(22) $\det(A^{-1}) = 1/\det(A)$ ...

(23) $\det(A^n) = \det(A)^n$ ...

(24) $\det(I + uv^T) = 1 + u^T v$ ...

6.2 Inverses

Might be reduced, unless cofactor matrices have a nice representation?

Chapter 7

Advanced Derivatives

In the Simple Derivatives chapter every network was built purely from contractions, so differentiation was just "delete the node". The remaining derivatives of the Matrix Cookbook involve a few genuinely nonlinear atoms: inverses, determinants, eigenvalues, norms, and structural constraints. The pleasant surprise is that each atom needs exactly *one* new local rule, and after applying it, delete-the-node does the rest.

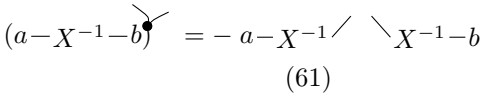
7.1 Derivatives of an Inverse

Differentiating both sides of $X^{-1}X = I$ with the product rule gives $\mathrm{d}(X^{-1})X + X^{-1}\mathrm{d}X = 0$, so $\mathrm{d}(X^{-1}) = -X^{-1}\,\mathrm{d}X\,X^{-1}$. In diagrams this is a *splitting rule*: to differentiate through an inverse node, split it into two inverse nodes with the new derivative edges dangling in between, and flip the sign:

Note the rule contains the plain matrix case: if X^{-1} appears once in any network, its derivative replaces the node by the split pair. The Matrix Cookbook's versions:

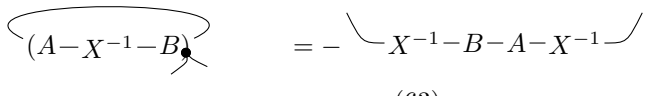
$$\frac{\partial Y^{-1}}{\partial x} = -Y^{-1}\frac{\partial Y}{\partial x}Y^{-1} \tag{59}$$

$$\frac{\partial a^T X^{-1} b}{\partial X} = -X^{-T}ab^T X^{-T}$$



(61)

$$\frac{\partial \mathrm{Tr}(AX^{-1}B)}{\partial X} = -(X^{-1}BAX^{-1})^T$$



(63)

$$\frac{\partial \mathrm{Tr}((X+A)^{-1})}{\partial X} = -\big((X+A)^{-1}(X+A)^{-1}\big)^T \tag{64}$$

In (61) and (63) we simply split the inverse node; the two dangling stubs are the row and column of the result, and reading them off gives the classical transposes. In (63), the trace loop connects the two split halves into the single chain $X^{-1}BAX^{-1}$.

When the inverse's *argument* contains X several times, the product rule adds one split term per occurrence. For B and C symmetric the Matrix Cookbook lists

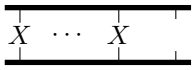
$$\frac{\partial}{\partial X}\mathrm{Tr}\big[(X^TCX)^{-1}A\big] \tag{125}$$
$$= -\big(CX(X^TCX)^{-1}\big)(A+A^T)(X^TCX)^{-1}$$
$$\frac{\partial}{\partial X}\mathrm{Tr}\big[(X^TCX)^{-1}(X^TBX)\big] \tag{126}$$
$$= -2CX(X^TCX)^{-1}X^TBX(X^TCX)^{-1} + 2BX(X^TCX)^{-1}$$
$$\frac{\partial}{\partial X}\mathrm{Tr}\big[(A+X^TCX)^{-1}(X^TBX)\big] \tag{127}$$
$$= -2CX(A+X^TCX)^{-1}X^TBX(A+X^TCX)^{-1} + 2BX(A+X^TCX)^{-1}$$

These all follow mechanically: delete the two X's of the inner X^TBX (the $+2BX(\cdots)^{-1}$ terms), and split the inverse around the two X's inside it (the $-2CX(\cdots)$ terms); the factors of 2 come from the symmetry of B and C pairing up the terms. We leave the diagram derivation of (126) as an exercise.

7.2 Derivatives of a Determinant

Recall from the previous chapter that $\det(X)$ is the closed diagram of n copies of X between two (normalized) anti-symmetrizer bars. The node X appears n times, so delete-the-node gives a sum of n networks with one copy removed — and by the symmetry of the bars, all n are equal:

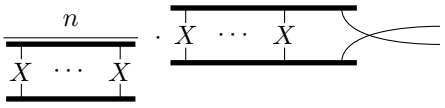
$$\frac{\partial \det(X)}{\partial X} \;=\; n \cdot$$



The diagram on the right — $n-1$ copies of X with one empty slot — is a matrix in the two dangling edges. Classically it is $\frac{1}{n}\,\mathrm{adj}(X)^T$, the (transposed, scaled) *adjugate*, and for invertible X it equals $\frac{1}{n}\det(X)\,X^{-T}$.

Penrose [23] read this identity the other way around, as a diagrammatic formula for the inverse itself:

$$X^{-1} \;=$$



where the two free legs of the empty slot *cross* on their way out: the crossing is exactly the transpose that turns the adjugate's X^{-T} into X^{-1}. So we recover the Matrix Cookbook identities

$$\frac{\partial \det(X)}{\partial X} = \det(X)\,(X^{-1})^T \tag{49}$$
$$\frac{\partial \det(Y)}{\partial x} = \det(Y)\mathrm{Tr}\Big[Y^{-1}\frac{\partial Y}{\partial x}\Big] \tag{46}$$

$$\frac{\partial \ln|\det(X)|}{\partial X} = (X^{-1})^T \tag{57}$$

$$\frac{\partial \det(AXB)}{\partial X} = \det(AXB)\,(X^{-1})^T \tag{51}$$

$$\frac{\partial \det(X^k)}{\partial X} = k\det(X^k)\,X^{-T} \tag{58}$$

where (46) is the chain-rule form of (49), (57) follows since $\mathrm{d}\ln\det = \mathrm{d}\det / \det$, and (51), (58) are exercises (in (58) the closed stack contains kn copies of X).

7.2.1 The Characteristic Polynomial

The determinant of $I+tA$ collects the *elementary symmetric polynomials* e_k of the eigenvalues of A, and each has a clean diagram: e_k is the closed stack of just k copies of A between bars of width k.

$$\det(I{+}tA) = \sum_{k=0}^{n} t^k e_k, \qquad e_1 = \quad \overset{\frown}{A} \quad , \qquad e_2 = \overline{\underline{A \quad A}} \;=\; \tfrac{1}{2}\big(\mathrm{Tr}(A)^2 - \mathrm{Tr}(A^2)\big), \quad \ldots$$

Expanding the width-2 bar as $\frac{1}{2}(\mathrm{id}-\mathrm{swap})$ gives the e_2 formula on the right. In particular, for $F(t) = \det(I + tA)$ we get $F'(0) = e_1 = \mathrm{Tr}(A)$ and $F''(0) = 2e_2 = \mathrm{Tr}(A)^2 - \mathrm{Tr}(A^2)$.

7.3 Derivatives of Eigenvalues

Two eigenvalue sums are derivatives we already know:

$$\frac{\partial}{\partial X}\sum_i \lambda_i = \frac{\partial}{\partial X}\mathrm{Tr}(X) = I \tag{65}$$

$$\frac{\partial}{\partial X}\prod_i \lambda_i = \frac{\partial}{\partial X}\det(X) = \det(X)X^{-T} \tag{66}$$

For a single eigenvalue, let X be symmetric with a simple eigenvalue λ and unit eigenvector v, so $\lambda = v^T X v$. Differentiating, the terms involving $\mathrm{d}v$ vanish: $\mathrm{d}\lambda = 2\lambda\, v^T \mathrm{d}v + v^T \mathrm{d}X\, v = \lambda\,\mathrm{d}(v^T v) + v^T \mathrm{d}X\, v$, and $v^T v = 1$ is constant. What remains is delete-the-node applied to v–X–v:

$$\frac{\partial \lambda}{\partial X} = vv^T \qquad\qquad (v{-}X{-}v) \;=\; v \diagup \diagdown v \tag{67}$$

The Matrix Cookbook also gives the eigenvector derivative $\partial v = (\lambda I - X)^{+}\partial(X)\, v$ (68), whose pseudo-inverse we will not attempt to draw here. As a corollary of (67), the spectral norm $\|X\|_2 = \sigma_1$ has gradient $u_1 v_1^T$, the outer product of the top singular vectors.

7.4 Derivatives of Norms

$$\frac{\partial \|x-a\|_2}{\partial x} = \frac{x-a}{\|x-a\|_2} \tag{129}$$

$$\frac{\partial}{\partial x}\frac{x-a}{\|x-a\|_2} = \frac{I}{\|x-a\|_2} - \frac{(x-a)(x-a)^T}{\|x-a\|_2^3} \tag{130}$$

$$\frac{\partial \|x\|_2^2}{\partial x} = 2x \qquad (x \text{—} x) = 2 \text{—} x \tag{131}$$

$$\frac{\partial \|X\|_F^2}{\partial X} = 2X \qquad (X \frown X) = 2 \diagdown X \diagup \tag{132}$$

The squared norms are quadratic, so delete-the-node handles them directly (X appears twice in $\mathrm{Tr}(X^T X)$, giving the factor 2). The unsquared (129) and its Jacobian (130) compose the scalar function $\sqrt{\cdot}$ with a quadratic network — exactly the chain-rule machinery of Chapter 4.

7.5 Derivatives of Structured Matrices

So far X had no special structure: every entry was a free parameter. If X is constrained — symmetric, diagonal, Toeplitz — the free parameters are fewer, and the honest way to differentiate is to write X as a linear function of its free parameters and use the chain rule. Diagrammatically, the constraint is one more tensor in the network: the *structure tensor* S with $X = S(\theta)$ for parameter vector (or matrix) θ, and the gradient with respect to θ is the unconstrained gradient $G = \partial f/\partial X$ pushed through S.

Symmetric. For symmetric X the independent parameters are the entries on and below the diagonal, and pushing G through the structure gives

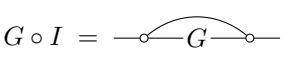
$$\frac{df}{dX} = G + G^T - G \circ I \tag{138}$$

$$\frac{\partial \mathrm{Tr}(AX)}{\partial X} = A + A^T - A \circ I \tag{139}$$

$$\frac{\partial \det(X)}{\partial X} = \det(X)\big(2X^{-1} - X^{-1} \circ I\big) \tag{140}$$

$$\frac{\partial \ln\det(X)}{\partial X} = 2X^{-1} - X^{-1} \circ I \tag{141}$$

The pattern $G + G^T$ is (twice) the symmetrization squiggle from the introduction; the correction $G \circ I$ removes the double-counted diagonal, drawn with copy tensors as

$G \circ I \;=$ 

(the arc between the two copy tensors forces the row and column index to agree, so only the diagonal of G survives). Note that (140) and (141) are just (49) and (57) pushed through the rule (138).

Diagonal. For diagonal $X = \mathrm{diag}(\theta)$ the structure tensor *is* the order-3 copy tensor, and the chain rule contracts G with it:

$$\frac{\partial \mathrm{Tr}(AX)}{\partial X} = A \circ I \tag{142}$$

Toeplitz. A Toeplitz matrix is constant along diagonals, one parameter per diagonal; the gradient with respect to the k-th parameter is the sum of G along the k-th diagonal (see the exercises).

7.6 Exercises

Exercise 45. Derive (126) with diagrams: draw $\mathrm{Tr}[(X^TCX)^{-1}(X^TBX)]$ as a closed chain containing four copies of X and one inverse node, apply delete-the-node to the two outer copies and the splitting rule to the two inner ones, and collect terms using the symmetry of B and C.

Exercise 46. Prove $\frac{\partial \det(X^{-1})}{\partial X} = -\det(X^{-1})(X^{-1})^T$ (62) by combining the determinant stack with the splitting rule.

Exercise 47. Prove the cofactor identity $\sum_k \frac{\partial \det(X)}{\partial X_{ik}} X_{jk} = \delta_{ij} \det(X)$ (47) by reinserting a copy of X into the empty slot of the adjugate diagram.

Exercise 48. Prove (58), $\partial \det(X^k)/\partial X = k \det(X^k) X^{-T}$: the stack for $\det(X^k)$ contains kn copies of X; deleting one and using adj gives the result.

Exercise 49. Let T be Toeplitz with parameters $t_{-n+1}, \ldots, t_{n-1}$, one per diagonal. Show that $\frac{\partial \mathrm{Tr}(A^TT)}{\partial t_k}$ is the sum of the entries of A along the k-th diagonal, and draw the structure tensor of T.

Exercise 50. Find the derivative of the Gaussian density

$$f(x, \Sigma, \mu) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

with respect to Σ, combining the splitting rule, the log-determinant (57), and the chain rule of Chapter 4.

Chapter 8

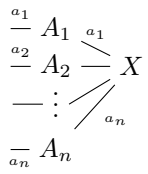
Special Tensors

Some tensors have special structure, other than rank/order, that give them nice properties, like fast contraction. Below are some examples.

8.1 Fast Kronecker Multiplication

Say we want to compute $(A_1 \otimes A_2 \otimes \cdots \otimes A_n)x$, where A_i is an $a_i \times a_i$ matrix, and $x \in \mathbb{R}^{a_1 a_2 \cdots a_n}$. If we first compute the Kronecker product, and then the matrix-vector multiplication, this would take $(a_1 \cdots a_n)^2$ time.

Instead we can reshape x into an $a_1 \times \cdots \times a_n$ tensor and perform the multiplication



by contracting the a_i edges one by one. This takes time

$$a_1^2(a_2 \cdots a_n) + a_2^2(a_1 a_3 \cdots a_n) + \cdots + a_n^2(a_1 \cdots a_{n-1}) = (a_1 + \cdots + a_n)(a_1 \cdots a_n),$$

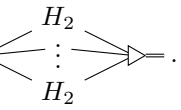
which is the basis of many fast algorithms as we will see.

8.2 Hadamard Transform

The Hadamard matrix is defined as $H_{2^n} = H_2^{\otimes n} = H_2 \otimes \cdots \otimes H_2$ where $H_2 = \left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$. For example

$$H_4 = H_2 \otimes H_2 = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

This gives a very simple tensor diagram representation as:

$H_{2^n} =$  .

The Fast Hadamard Transform (FHT) is usually described recursively by:

$$H_{2^n} x = \begin{bmatrix} H_{2^{n-1}} & H_{2^{n-1}} \\ H_{2^{n-1}} & -H_{2^{n-1}} \end{bmatrix} \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix},$$

where $\left[\begin{smallmatrix} x^{(1)} \\ x^{(2)} \end{smallmatrix}\right]$ is the first and second half of x. Because of the redundancy in the matrix multiplication (it only depends on $H_{2^{n-1}} x^{(1)}$ and $H_{2^{n-1}} x^{(2)}$), the algorithm computes $H_N x$ in $O(N \log N)$ time.

Alternatively we could just use the general fact, as described above, where $a_i = 2$ for all i. Then the "fast Kronecker multiplication" method takes time $(a_1 a_2 \cdots a_n)(a_1 + a_2 + \cdots + a_n) = 2N \log_2 N$.

8.3 Fourier Transform

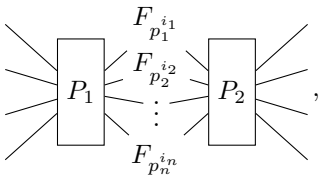
The Discrete Fourier Matrix is defined by $(F_N)_{j,k} = \omega^{jk}$, where $\omega = e^{-2\pi i/N}$:

$$F_N = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}.$$

Using the function syntax from earlier, and assuming n is the vector $[0, 1, \ldots, N-1]$, an alternative way to write this would be:

$$\exp(-2\pi i \cdot /N) \leftarrow \begin{matrix} n \\ n \end{matrix}$$

The Good-Thomas Fast Fourier Transform (FFT) uses a decomposition based on the Chinese Remainder Theorem:

$F_N =$  ,

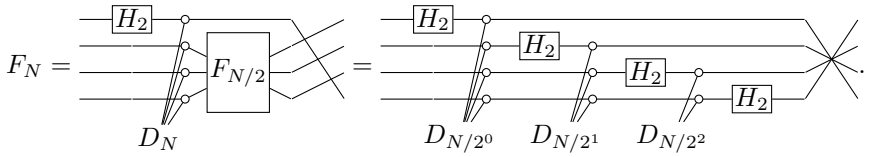
where $N = p_1^{i_1} p_2^{i_2} \cdots p_n^{i_n}$ is the prime factorisation of N, and P_1 and P_2 are some permutation matrices.

Using fast Kronecker multiplication, this takes $(p_1^{i_1} + \cdots + p_n^{i_n})N$ time. By padding x with zeros, we can increase N by a constant factor to get a string of $n = O(\log(N)/\log\log(N))$ primes, the sum of which is $\sim n^2 \log n = O(\log(N)^2/\log\log N)$. The complete algorithm thus takes time $O(N \log(N)^2)$. Next we will see how to reduce this to $O(N \log N)$.

The classical Cooley-Tukey FFT algorithm uses a recursion:

$$F_N = \begin{bmatrix} I & I \\ I & -I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & D_{N/2} \end{bmatrix} \begin{bmatrix} F_{N/2} & 0 \\ 0 & F_{N/2} \end{bmatrix} \begin{bmatrix} \text{even-odd} \\ \text{permutation} \end{bmatrix},$$

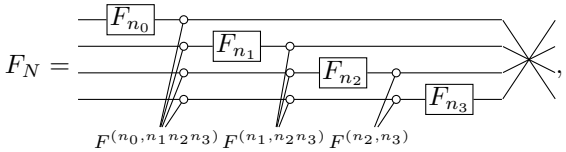
where $D_{N/2} = \mathrm{diag}(1, \omega, \omega^2, \ldots, \omega^{N/2-1})$ is the diagonal matrix of "twiddle factors", with $\omega = e^{-2\pi i/N}$ as before. The even-odd permutation moves all the even values to the start. If we reshape the length-2^n index space into $2 \times \cdots \times 2$, this permutation is just $P_N =$  , or in pytorch: `x.permute([3,0,1,2])`. Also note that $\left[\begin{smallmatrix} I & I \\ I & -I \end{smallmatrix}\right] = H_2 \otimes I$ and $\left[\begin{smallmatrix} F_{N/2} & 0 \\ 0 & F_{N/2} \end{smallmatrix}\right] = I_2 \otimes F_{N/2}$. In the diagrams below we write D_N for the full length-N diagonal matrix $\left[\begin{smallmatrix} I & 0 \\ 0 & D_{N/2} \end{smallmatrix}\right] = I_{N/2} \oplus D_{N/2}$. So we can write in tensor diagram notation:



Since one can multiply with the permutation and diagonal matrices in linear time, the $O(N \log N)$ time complexity follows from the same argument as for Hadamard.

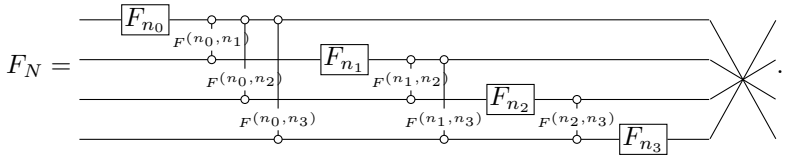
Note there are a bunch of symmetries, such as by transposing (horizontal flip), since the matrix is symmetric. Or by pushing the permutation to the left side.

We don't have to split the matrix in half, we can also split it in thirds, fourths, etc. With this generalized Cooley-Tukey algorithm, we get the following diagram:



where $n_0 n_1 n_2 n_3 = N$. Here we replaced the D_N matrix with the "generalized" Fourier matrix $F^{(a,b)}$ matrix, which is defined as $F^{(a,b)}_{j,k} = e^{-2\pi ijk/(ab)}$, and we reshaped as necessary. In the simple case of where we split in just two parts of roughly $n_1 = n_2 = \sqrt{N}$, this is also called "Four step FFT" or Bailey's FFT algorithm.

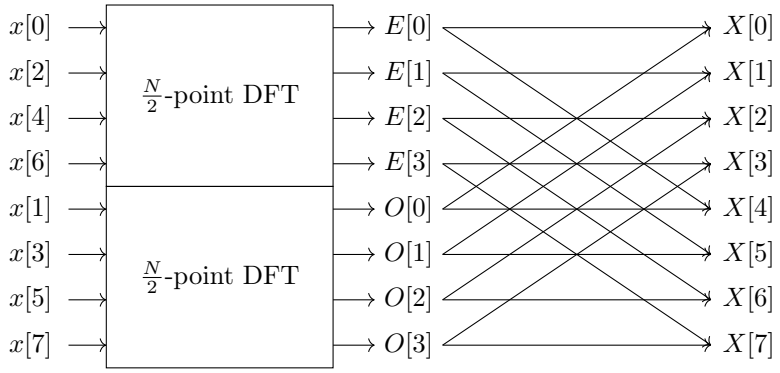
We can use the property $F^{(a,bc)} = F^{(a,b)} \bullet (F^{(a,c)})^{1/b}$ to simplify the diagram further. Here $\bullet$ denotes the join over the shared row index j: reshaping the column index k into a pair (k_1, k_2), we set $(F^{(a,b)} \bullet (F^{(a,c)})^{1/b})_{j,(k_1,k_2)} = F^{(a,b)}_{j,k_1} \big((F^{(a,c)})^{1/b}\big)_{j,k_2}$, where the elementwise power $1/b$ is taken using the principal branch. This gives:



In the simple case where we split in 2 every time, this is also called the "Quantum Fourier Transform (QFT)" algorithm.

We hid some stuff above, namely that the matrices should be divided by different Ns.

Note that this figure may look different from some FFT diagrams you have seen. These typically look like this:



and have 2^n rows. The tensor diagram only has n rows (or $\log_2 N$).

8.3.1 Multi-dimensional Fourier Transform

This is just taking the Fourier transform along each axis.

8.3.2 Vandermonde Matrices

We can define the "Vandermonde Function", $\mathrm{vm}_N(x) = (1, x, x^2, \ldots, x^{N-1})$. Then the Vandermonde Matrix, $V_{i,j} = x_i^j$ is defined:

$$— \text{ vm } \leftarrow x —$$

If $a \in \mathbb{R}^N$ is a vector with the coefficients of the polynomial $P(x) = a_0 + a_1 x + \ldots$, we can get the "batch evaluation" vector, $(P(x_1), P(x_2), \ldots)$ as:

$$a – \text{ vm } \leftarrow x —$$

This can be done fast (in $O(n(\log n)^2)$ time) using the FFT described above. The idea is to define two polynomials that are zero in respectively the first and second half of the points:

$$m_0(x) = (x - x_1) \cdots (x - x_{n/2}) \quad \text{and}$$
$$m_1(x) = (x - x_{n/2+1}) \cdots (x - x_n).$$

We then compute $R_0 = P \bmod m_0$ and $R_1 = P \bmod m_1$ using fast polynomial division, which can be done in $O(n \log n)$ time using a FFT. This means

$$P(x) = Q_0(x) m_0(x) + R_0(x) \quad \text{and}$$
$$P(x) = Q_1(x) m_1(x) + R_1(x)$$

by construction, where R_0 and R_1 are polynomials of degree less than $n/2$. Because of how m_0 and m_1 were defined, we have

$$\begin{aligned} R_0(x_i) &= P(x_i) \quad \text{for } i \le n/2 \quad \text{and} \\ R_1(x_i) &= P(x_i) \quad \text{for } i > n/2. \end{aligned}$$

Thus to compute P on all n of the x_i, it suffices to compute the smaller polynomials R_0 and R_1 on each half of the points. (The Q_0 and Q_1 polynomials can be ignored.) This gives us a divide-and-conquer algorithm with $T(n) = 2T(n/2) + n \log n$, which implies $T(n) = O(n(\log n)^2)$.

8.4 Block matrices

Stuff like Schur complements is interesting. But can we say anything useful using tensor diagrams?

From the Matrix Cookbook: Regard the matrix

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

The Schur complement of block A_{11} of the matrix above is the matrix

$$C_2 = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

The Schur complement of block A_{22} of the matrix above is the matrix

$$C_1 = A_{11} - A_{12}A_{22}^{-1}A_{21}$$

Using the Schur complement, one can rewrite the inverse of a block matrix

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]^{-1} = \left[\begin{array}{c|c} I & 0 \\ \hline -A_{22}^{-1}A_{21} & I \end{array}\right] \left[\begin{array}{c|c} (A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1} & 0 \\ \hline 0 & A_{22}^{-1} \end{array}\right] \left[\begin{array}{c|c} I & -A_{12}A_{22}^{-1} \\ \hline 0 & I \end{array}\right]$$

The Schur complement is useful when solving linear systems of the form

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] = \left[\begin{array}{c} b_1 \\ b_2 \end{array}\right]$$

which has the following equation for x_1

$$(A_{11} - A_{12}A_{22}^{-1}A_{21})x_1 = b_1 - A_{12}A_{22}^{-1}b_2$$

When the appropriate inverses exists, this can be solved for x_1 which can then be inserted in the equation for x_2 to solve for x_2.

8.5 Symmetric, Skew-symmetric/Antisymmetric

Could introduce Penrose's symmetric tensors here?

8.6 Toeplitz Matrices

Could talk about the convolution tensor here...

8.7 Units, Permutation and Shift

Not that interesting...

Chapter 9

Decompositions

9.1 Higher-order singular value decomposition

Say we have an order n tensor A. We "unfold" A along each dimension. This means pulling the edge i to the left, and flattening the rest to the right. Then we compute the SVD, USV^T. Here U is a square matrix, which we keep. We multiply the ith edge of A by U^T (since U is orthogonal, U^T is also the inverse of U). The result is a "core" tensor as well as a sequence of U tensors. If we want a more compact SVD, we can make each U low rank, like normal SVD. There is also the "Interlacing computation" where we multiply the U^Ts onto A as we go along.

The resulting format — a core tensor contracted with a factor matrix on each mode — is called a "Tucker decomposition" (introduced by Tucker for order 3 tensors); the algorithm above for computing it is known as the higher-order SVD (HOSVD).

A related but different format is the CP decomposition (CANDECOMP/PARAFAC), which writes the tensor as a sum of r outer products of vectors. Here r may exceed the dimensions of the tensor, and the factors are in general neither orthogonal nor square. Finding the smallest such r — the tensor rank — is called tensor rank decomposition. If we were good at that, we could use it to factor $I^{\otimes 3}$ to get better matrix multiplication algorithms. Unfortunately tensor rank decomposition is NP hard.

Note that a Tucker decomposition does yield a rank decomposition of sorts: expanding the dense core tensor into its nonzero entries gives a sum of up to $\prod_i R_i$ outer products. It just won't be a minimal one.

9.2 Rank Decomposition

9.2.1 Border Rank

The border rank of a tensor is the smallest r such that the tensor is a limit of tensors of rank at most r. The border rank can be strictly smaller than the rank. For example, if x and y are linearly independent vectors, the tensor

$$T = x \otimes x \otimes y + x \otimes y \otimes x + y \otimes x \otimes x$$

has rank 3, but border rank 2. To see this, expand

$$(x + \varepsilon y)^{\otimes 3} = x \otimes x \otimes x + \varepsilon \left(x \otimes x \otimes y + x \otimes y \otimes x + y \otimes x \otimes x \right) + O(\varepsilon^2),$$

so

$$T = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon} \left[(x + \varepsilon y)^{\otimes 3} - x^{\otimes 3} \right],$$

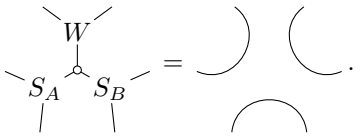
where each tensor in the limit is a sum of just two simple tensors.

9.3 Fast Matrix Multiplication

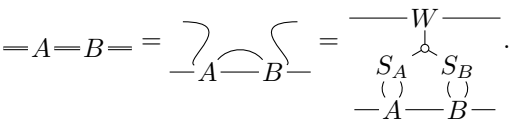
Strassen defines 3 tensors of shape $7 \times 2 \times 2$:

$$\begin{aligned}
S_A &= \left[\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 0 \\ 1 & 1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 1 & 1 \\ 0 & 0 \end{smallmatrix} \right], \left[\begin{smallmatrix} -1 & 0 \\ 1 & 0 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 1 \\ 0 & -1 \end{smallmatrix} \right] \right] \\
S_B &= \left[\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 1 \\ 0 & -1 \end{smallmatrix} \right], \left[\begin{smallmatrix} -1 & 0 \\ 1 & 0 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 1 & 1 \\ 0 & 0 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 0 \\ 1 & 1 \end{smallmatrix} \right] \right] \\
W &= \left[\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 1 \\ 0 & -1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 0 \\ 1 & 1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 1 & 1 \\ 0 & 0 \end{smallmatrix} \right], \left[\begin{smallmatrix} -1 & 0 \\ 1 & 0 \end{smallmatrix} \right], \left[\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix} \right], \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix} \right] \right]
\end{aligned}$$

These tensors have the neat property that they factor $I_2 \otimes I_2 \otimes I_2$:

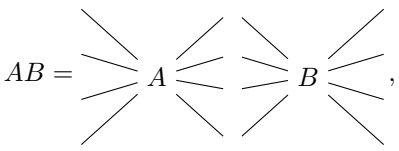


To multiply two matrices, A and B, faster than the normal n^3 time, we reshape them as block matrices, shape $(2, \frac{n}{2}, 2, \frac{n}{2})$ and use Strassen's tensor:



Contracting the edges in the right order, uses only $7/8\, n^3 + O(n^2)$ operations. The key is that the hyperedge of size 7 must *not* be contracted first. Instead we first contract the pairs of edges between S_A and A, and between S_B and B; each of these takes $7 \cdot 2^2 (n/2)^2 = O(n^2)$ operations. Next we contract $S_A A - S_B B$, a batched product over the hyperedge, which takes $7(n/2)^3 = \frac{7}{8} n^3$ operations. Finally we contract the result with W, which takes $2^2 \cdot 7(n/2)^2 = O(n^2)$ operations. The important term is the cubic $\frac{7}{8} n^3$, which, if instead computed recursively, leads to the classical $O(n^{\log_2 7})$ algorithm.

If we instead reshape to $(2, 2, \ldots, 2)$,



and using Strassen's tensor along each axis reduces the work by $(7/8)^{\log_2(n)}$, giving us matrix multiplication in time $n^{3+\log_2(7/8)} = n^{2.80735}$.

9.3.1 Other

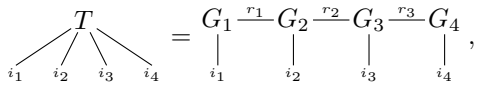
If we instead wrote A and B using (n, m) and (m, p) shaped blocks, we could factor $I_n \otimes I_m \otimes I_p$ and get a matrix multiplication algorithm using the same approach as the Strassen $(2, 2, 2)$ tensors above. Lots of papers have investigated this problem, which has led to the best algorithms by Josh Alman and others. For example, DeepMind found a rank 47 factorization of $I_3 \otimes I_4 \otimes I_5$.

Maybe a more interesting example is the $(4, 4, 4)$ tensor, for which they find a rank 47 factorization. An easy way to create a rank 49 is to take Strassen and double it. Would this be a nice thing to show? Maybe too messy? Well, actually their rank 47 construction only works in the "modular" case. Then $(3, 4, 5)$ is general.

Chapter 10

Tensor Trains

An order n tensor with all dimensions equal to d has d^n entries. For $d = 2$ and $n = 50$ that is already more numbers than any computer wants to store. The Tensor Train (TT) format [22] sidesteps this by writing the tensor as a chain – a "train" – of small order-3 tensors:



or in index notation,

$$T_{i_1,i_2,\ldots,i_n} = \sum_{r_1,\ldots,r_{n-1}} (G_1)_{i_1,r_1} (G_2)_{r_1,i_2,r_2} \cdots (G_n)_{r_{n-1},i_n}.$$

The tensors G_k are called the *cores* (or, sticking with the metaphor, the carriages) of the train. The dangling edges $i_1, \ldots, i_n$ are the original axes of T, and the internal edges $r_1, \ldots, r_{n-1}$ are new summation indices whose dimensions we get to choose. The smallest dimensions $(r_1, \ldots, r_{n-1})$ for which an exact representation exists are called the *TT-ranks* of T. If all TT-ranks are bounded by R, the format stores at most ndR^2 numbers instead of d^n.

In quantum physics, where the format was invented first, it is known as a *Matrix Product State* (MPS) [35, 28]. There the dangling edges are called "physical" dimensions and the internal edges "bond" dimensions. The name Matrix Product State comes from the observation that each entry of T is a product of matrices: if we fix the index i_k of each core, the slice $(G_k)_{:,i_k,:}$ is an $r_{k-1} \times r_k$ matrix, so

$$T_{i_1,\ldots,i_n} = (G_1)_{i_1,:} (G_2)_{:,i_2,:} \cdots (G_n)_{:,i_n}$$

is a product of a row vector, $n-2$ matrices, and a column vector, which we can evaluate in just $O(nR^2)$ time. This is the first hint of why the format is useful: we never need to materialize T to work with it.

Note that tensor trains generalize the two decompositions we have already seen. For $n = 2$ a tensor train is just a low-rank factorization $T = G_1 G_2$, and the TT-rank is the matrix rank. And if we force all internal edges to be copies of each other (contract them with a copy tensor), we recover the CP decomposition. Compared to the Tucker decomposition of Section 9.1, which stores a dense R^n core, the tensor train breaks the core itself into a chain, so the storage stays linear in n.

10.1 Cheap Operations on Trains

The point of a compressed format is that we can compute with it directly. Here are the most important operations. Throughout, T and T' are order n tensors, all of whose dimensions are d, given in TT format with all ranks bounded by R.

Inner products and norms. The inner product $\langle T, T' \rangle$ is the tensor network where all corresponding dangling edges are contracted:

$$\langle T, T' \rangle = \begin{array}{cccccccc} G_1 & - & G_2 & - & G_3 & - & G_4 \\ {\scriptstyle i_1}\big| & & {\scriptstyle i_2}\big| & & {\scriptstyle i_3}\big| & & {\scriptstyle i_4}\big| \\ G'_1 & - & G'_2 & - & G'_3 & - & G'_4 \end{array} .$$

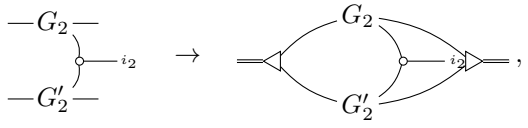
As we saw in Section 12.1, the cost of evaluating such a network depends on the contraction order. The right order here is to zip from left to right: first contract G_1 with G'_1, giving an $R \times R$ matrix on the two bond edges; then absorb G_2, then G'_2, and so on. Every intermediate tensor has at most 3 edges, and the total cost is $O(ndR^3)$. (Contracting all the bottom row first, by comparison, would create an intermediate tensor with n dangling edges – exactly the object we are trying to avoid.) Taking $T' = T$ gives the Frobenius norm $\|T\|_F^2$ at the same cost.

Sums. The sum $T + T'$ is again a tensor train, with ranks at most $R + R'$: simply take the new cores to be block diagonal,

$$G''_k[i_k] = \begin{pmatrix} G_k[i_k] & 0 \\ 0 & G'_k[i_k] \end{pmatrix},$$

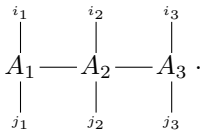
where $G_k[i_k]$ denotes the i_kth matrix slice as above, and where the first and last cores are concatenated as a row/column of blocks instead. Multiplying the block matrices together telescopes into $T_{i_1,\ldots,i_n} + T'_{i_1,\ldots,i_n}$, as promised (Exercise 51).

Hadamard products. The entry-wise product $T \circ T'$ connects each pair of dangling edges with an order-3 copy tensor, as in Section 3.6. Grouping each G_k with G'_k and the copy tensor between them gives new cores whose bond edges are the *pairs* (r_k, r'_k), which we can bundle with the flattening triangle from the Kronecker chapter:



so $T \circ T'$ has TT-ranks at most RR'. The ranks multiply rather than add, which is why repeated Hadamard products (say, when representing probability distributions or running many gradient steps) require periodic re-compression – see the rounding procedure below.

Matrices in TT format. A matrix $A \in \mathbb{R}^{d^n \times d^n}$ can be stored as an order $2n$ tensor where core k carries *two* dangling edges, one row-index i_k and one column-index j_k:

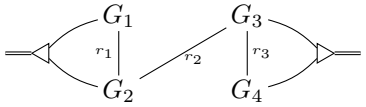


This is called a TT-matrix, or in physics a *Matrix Product Operator* (MPO) [21]. The product $y = Ax$ of an MPO (ranks S) with a tensor train x (ranks R) is computed by stacking the two chains and contracting the shared edges; the result is again a tensor train, with ranks at most RS, computable in time $O(nd^2R^2S^2)$. This is the trick behind "tensorizing" neural networks [20]: a 1024×1024 dense layer, reshaped as an order-20 tensor of dimensions 2 and stored as a low-rank MPO, can use hundreds of times fewer parameters, while the matrix-vector product – and, by the results of the derivative chapters, backpropagation – runs directly in the compressed format.

10.2 What are the TT-ranks?

For matrices we understand rank well. Happily, TT-ranks reduce to matrix ranks. Cut the train between carriage k and $k+1$, and let $T_{\langle k \rangle} \in \mathbb{R}^{d^k \times d^{n-k}}$ denote the flattening of T that bundles the first k edges into a row index and the rest into a column index.

The train itself proves that $\operatorname{rank}(T_{\langle k \rangle}) \le r_k$: everything to the left of bond r_k is a $d^k \times r_k$ matrix (after flattening), everything to the right is $r_k \times d^{n-k}$, and $T_{\langle k \rangle}$ is their product:



(Here we cut at $k = 2$, redrew the chain so that carriages $1, 2$ sit on the left of the bond edge r_2 and carriages $3, 4$ on the right, and flattened both sides.)

More surprisingly, the bound is tight, so

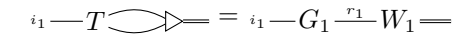
$$\text{TT-rank}_k(T) = \operatorname{rank}\left(T_{\langle k \rangle}\right),$$

and the minimum is achieved for all k *simultaneously* by a single train [22]. We prove this constructively in the next section.

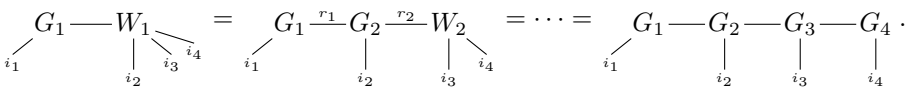
The same cut argument works for any tensor network, not just trains, and is worth stating in general [26]: *if a tensor T is given by any tensor network, and we split its dangling edges into two groups I and J, then the rank of the flattening $T_{(I;J)}$ is at most the product of the dimensions of the network edges crossed by any cut separating the I-edges from the J-edges.* The network factors through the cut, just like the picture above. For example, this immediately gives $\operatorname{rank}(AB) \le$ (dimension of the middle edge), and it explains why networks with loops, like the PEPS grids used in physics, can represent tensors whose flattenings have exponentially large rank even when every core is small: a cut may have to cross many edges at once.

10.3 TT-SVD: Computing the Decomposition

Unlike the CP decomposition, which is NP-hard to compute, an exact (or optimally truncated, in the SVD sense) tensor train can be computed by a sequence of $n-1$ matrix factorizations. The algorithm peels off one carriage at a time. Start by flattening all but the first edge of T and computing a (thin) SVD or QR factorization:

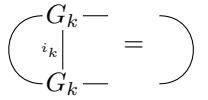


where G_1 has orthonormal columns (it is the U of the SVD) and $W_1 = \Sigma V^T$ is whatever remains. Now unflatten W_1, group its bond edge r_1 together with the edge i_2, flatten the rest, and factor again:



Since each step is an exact factorization, the final train is exact, and the rank produced at step k is $\operatorname{rank}(T_{\langle k \rangle})$ – which proves the theorem of the previous section. If instead we *truncate* each SVD, keeping only the largest singular values, we get a low-rank approximation whose error is controlled: truncating with error ε_k at step k gives $\|T - \widetilde{T}\|_F^2 \le \sum_k \varepsilon_k^2$, which is within a factor $\sqrt{n-1}$ of the best possible TT approximation at those ranks [22].

Orthogonal cores. The cores produced by TT-SVD are special: each G_k (except the last) satisfies



that is, flattening the left bond and the dangling edge together gives a matrix with orthonormal columns. Such cores are called *left-orthogonal*. Orthogonality is what makes the truncated SVD steps above optimal, and it is also numerically pleasant: if all cores but one are orthogonal (a "canonical form", with the remaining core called the center of orthogonality), then the Frobenius norm of the whole train equals the norm of that single core, and truncating there is exactly a local SVD problem. Algorithms on trains, like the alternating least squares (ALS/DMRG) methods that optimize one core at a time [13, 28], sweep the center of orthogonality back and forth along the train using cheap QR factorizations.

Rounding. We saw that sums and Hadamard products increase the TT-ranks. The fix is *TT-rounding* [22]: given a train with inflated ranks R', sweep right-to-left making all cores right-orthogonal (QR factorizations only), then sweep left-to-right applying truncated SVDs as in TT-SVD. This costs $O(ndR'^3)$ – polynomial in everything – and is exactly equivalent to running TT-SVD on the full tensor, without ever materializing it.

10.4 Quantized Tensor Trains

Nothing stops us from applying tensor trains to plain vectors. Take a vector $v \in \mathbb{R}^{2^n}$, reshape it into an order n tensor of shape $2 \times 2 \times \cdots \times 2$ (index it by the bits of the

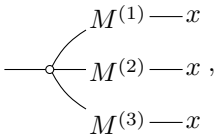
original index), and decompose. This is called the *quantized* tensor train (QTT) [21]. If the TT-ranks come out bounded by R, we have compressed v from 2^n numbers down to $O(nR^2)$.

Remarkably many structured vectors have tiny QTT ranks. The geometric sequence $v_i = q^i$ has rank 1 (Exercise 52), the sequences $v_i = \sin(ai + b)$ and $v_i = i$ have rank 2, and polynomials of degree p have rank $p + 1$. Structured matrices compress too, as TT-matrices on bit indices: the identity is rank 1 (the QTT cores are copy tensors), the tridiagonal second-difference (Laplacian) matrix has rank 3, and the DFT matrix from the Fourier chapter has QTT ranks polylogarithmic in the accuracy $1/\varepsilon$ after ε-truncation – the butterfly diagrams we drew there are morally a tensor train growing one carriage per FFT level. This gives algorithms with log-dimension complexity: solving a discretized PDE on 2^{50} grid points becomes a sequence of operations on trains of 50 tiny cores.

10.5 Sketching Tensor Products

We finish the chapter with a different way random compression interacts with tensor networks, following [1]. Suppose we want the rank-1 tensor $x^{\otimes p} = x \otimes x \otimes \cdots \otimes x \in \mathbb{R}^{d^p}$ – for instance because its inner products $\langle x^{\otimes p}, y^{\otimes p} \rangle = \langle x, y \rangle^p$ compute the polynomial kernel in machine learning. Materializing d^p entries is out of the question, so we want a random linear *sketch* $\Pi \in \mathbb{R}^{m \times d^p}$, with small m, such that $\|\Pi x^{\otimes p}\|_2 \approx \|x^{\otimes p}\|_2$ (and more generally, such that inner products are preserved). The catch: Π itself has md^p entries, so Π must *itself* be a tensor network we never materialize.

A first attempt. The simplest idea is to take p independent random sketch matrices $M^{(1)}, \ldots, M^{(p)} \in \mathbb{R}^{m \times d}$ and combine them row-wise, so that row j of Π is the tensor product of the jth rows of the $M^{(k)}$. Row-wise combination is exactly a copy tensor on the output edge (this is the Khatri–Rao product; compare Section 3.6), so the sketched vector is

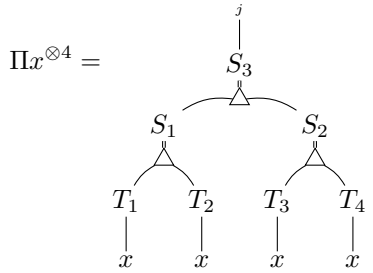
$\Pi x^{\otimes p} =$ 

,

i.e. $(\Pi x^{\otimes p})_j = \prod_{k=1}^{p} (M^{(k)} x)_j$, computable in $O(p\,m\,d)$ time – no d^p anywhere. Unfortunately, the products of p independent random variables appearing in each coordinate have heavy tails: one can show that preserving norms requires m to grow *exponentially* in p (roughly as 3^p), even when each $M^{(k)}$ is a perfectly good JL matrix [1]. The same exponential dependence afflicts TensorSketch [25], the FFT-based version of this construction.

Recursive sketching. The fix of [1] is to build Π as a *tree* of small sketches. Take base sketches $T_k \in \mathbb{R}^{m \times d}$ (one per leaf) and $S_k \in \mathbb{R}^{m \times m^2}$ (one per internal node), and sketch

pairwise, recursively:



Reading from the bottom: each leaf compresses one copy of x to m dimensions; each internal node takes two m-dimensional sketches, forms (implicitly) their m^2-dimensional tensor product, and compresses it back to m dimensions. The whole sketching matrix Π is a tree tensor network with $2p-1$ random cores, each of them small. With this construction the exponential blow-up disappears: a target dimension m that is *polynomial* in p and $1/\varepsilon$ suffices, and the analysis shows the error compounds only additively over the $O(p)$ nodes of the tree [1].

Two practical notes. First, the internal sketches S_k must support fast multiplication on tensor products $u \otimes v$ *without* forming the m^2-dimensional vector; degree-two TensorSketch does this in $O(m \log m)$ time via the FFT, so the full sketch takes $O(p(\mathrm{nnz}(x) + m \log m))$ time. Second, the shape of the tree is a design choice, just like a contraction order: a balanced tree has depth $\log_2 p$, while a maximally unbalanced "caterpillar" tree – absorb one new leaf at each level – makes Π literally a tensor train whose carriages are random sketches. Random tensor networks, in other words, are not just objects we compress; they are also the compressors.

Because the sketch is linear in $x^{\otimes p}$, it extends beyond powers of a single vector: it applies to any vector in $\mathbb{R}^{d^p}$, and by Taylor-expanding $e^{\langle x,y\rangle} = \sum_p \langle x, y\rangle^p / p!$ one gets sketches for the Gaussian kernel $e^{-\|x-y\|^2/2}$ whose dimension is polynomial in the data radius and logarithmic in the accuracy – see [1] for details, and the Taylor material in Chapter 4 for the diagrammatic view of such expansions.

10.6 Exercises

Exercise 51. Verify the block construction for sums: with cores $G''_k[i] = \left[\begin{smallmatrix} G_k[i] & 0 \\ 0 & G'_k[i] \end{smallmatrix}\right]$ for $1 < k < n$, $G''_1[i] = \left[\begin{smallmatrix} G_1[i] & G'_1[i] \end{smallmatrix}\right]$ and $G''_n[i] = \left[\begin{smallmatrix} G_n[i] \\ G'_n[i] \end{smallmatrix}\right]$, show that the resulting train represents $T + T'$.

Exercise 52. Let $v \in \mathbb{R}^{2^n}$ be the geometric sequence $v_i = q^i$, $i = 0, \ldots, 2^n - 1$. Show that the QTT representation of v has all ranks equal to 1. (Hint: write i in binary as $i = \sum_k b_k 2^{n-k}$ and use $q^i = \prod_k (q^{2^{n-k}})^{b_k}$.) Then show that $v_i = \sin(ai + b)$ gives ranks at most 2. (Hint: angle addition formulas, or write sin using two complex geometric sequences.)

Exercise 53. Show that the order n copy tensor has all TT-ranks equal to d, with cores that are themselves order-3 copy tensors. In pictures: a hyperedge connecting n tensors can always be replaced by a chain of 3-way copy tensors.

Exercise 54. Use the cut bound to show that the TT-ranks of an order n tensor satisfy $r_k \leq \min(d^k, d^{n-k})$, and that both the all-ones tensor and, more generally, any rank-1 tensor $a_1 \otimes \cdots \otimes a_n$ has all TT-ranks 1.

Exercise 55. Prove diagrammatically that the first-attempt sketch satisfies $(M^{(1)} \bullet M^{(2)})(x \otimes y) = (M^{(1)}x) \circ (M^{(2)}y)$, where $\bullet$ is the row-wise (Khatri–Rao) product and $\circ$ the Hadamard product.

Chapter 11

Machine Learning Applications

11.1 Least Squares

11.2 Hessian of Cross Entropy Loss

11.3 Convolutional Neural Networks

11.4 Transformers / Attention

11.5 Tensor Sketch

11.6 Reinforcement Learning

When considering imperfect information games, let's try to approximate each player's strategy with a low rank tensor. We can use the tensor sketch to approximate the tensor. Basically that means picking a random outer product of vectors and multiplying on the derivative. This should be fast...

Chapter 12

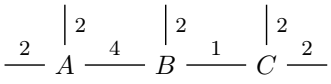
Tensor Algorithms

12.1 Tensor Contraction Orders

Throughout the previous chapters, we drew a *lot* of tensor networks. While the theoretical aspect is sufficient for some applications, there are indeed research areas, mostly computational ones such as quantum circuit simulation, where one wants to actually compute the tensor underlying the tensor network. This is done by the so-called *tensor contractions*. To see what this means, consider the following 2-tensor diagram, where we contract two tensors A and B:

Note that we also write the sizes of the respective dimensions, i.e., A is a $2 \times 2 \times 4$-tensor, while B is a 4×2-matrix. Now, the *contraction cost* is defined as the number of FLOPS performed during the contraction; as a convention, this is the number of scalar multiplications.[1] In our example, the contraction cost is $2 \times 2 \times 4 \times 2 = 32$, i.e., we simply multiply the dimension sizes.

The previous example was rather small. However, tensor networks in the wild tend to have hundreds of tensors. Naturally, these also need to be contracted to a single tensor. Now comes the interesting part: The *order* in which we perform these contraction can have a tremendous impact on the execution time. To get an intuition for this, consider an extended example of a 3-tensor diagram:



If we were to contract A with B first (and the resulting tensor with C), we would have a cost of $2^2 \times 4 \times 2 + 2^3 \times 1 \times 2^2 = 32 + 32 = 64$ multiplications, whereas performing the contraction between B and C at the beginning would result in $2^3 \times 1 \times 2^2 + 2^2 \times 4 \times 2^3 = 32 + 128 = 160$ multiplications in total. Hence, the first order is much better.

It is thus natural to ask: *Can we always find the optimal contraction order?*

[1] This assumes a naive implementation of the operation, i.e., nested loops over the dimensions.

12.1.1 Algorithms

We summarize well-known results and frameworks to find good contraction orders.

Optimal Algorithms

Indeed, finding the *optimal* contraction order for arbitrary tensor network shapes is pretty hard; better said, NP-hard [4]. There is a well-known exact algorithm running in $O(3^n)$-time [24], where n is the number of tensors in the network. This finds the optimal contraction order with respect to the total contraction cost. If one is interested in minimizing the size of the largest intermediate tensor, i.e., to optimize for the memory used during the execution, this can be done faster in $O(2^n n^3)$-time [32].

The good news is that for some restricted shapes of tensor networks, there are indeed efficient algorithms. A classic example is that dynamic programming solution for the matrix-chain problem [5], which is just *our* problem, but only for matrices. The naive algorithm runs in $O(n^3)$-time, but can be implemented in $O(n^2)$-time [37] (or even in $O(n \log n)$-time [14, 15]). Another shape for which polynomial-time algorithms exist is that of *tree* tensor networks [36, 33].

Another prominent way to optimize contraction orders is via the tree decomposition of the *line graph* representation of the tensor network [18, 7, 29]. In particular, this results in a contraction order with a maximal intermediate tensor rank bounded by the width of the tree decomposition (plus one) [18]. Loosely speaking, the treewidth — the smallest width over all tree decompositions of a graph — measures how tree-like the graph is. This does not directly solve our problem since finding the tree decompositions of the smallest width is itself hard [17]. Two well-known frameworks to find good tree decompositions are `QuickBB` [8] and `FlowCutter` [34, 11].

Best-Effort Algorithms

However, once we want to contract *arbitrary* network shapes, the best we can do is to fall back on heuristics or approximations. Two well-known frameworks are `opt_einsum` [30] and `cotengra` [9], which aim to optimize the contraction order (also referred to as "contraction path") of arbitrary einsums: For tensor networks where the optimal algorithm would be too slow, `opt_einsum` applies an ad-hoc greedy algorithm, while `cotengra` uses hypergraph partitioning [27], along with a Bayesian optimization approach, which has been later refined [31]. Other algorithms adopted from database query optimization are implemented in `netzwerk` [33]. Another option is to *learn* the best contraction orders, e.g., using reinforcement learning [19].

Naturally, the above heuristics do not come with an optimality guarantee. There exists a $(1+\varepsilon)$-approximation $O^*(2^n/\varepsilon)$-time algorithm that minimizes the sum of the intermediate tensor sizes [32].

Worth mentioning is the SQL-view of einsum [3]: It allows to perform the entire tensor network contraction on any database engine. Indeed, for some einsum classes, running the contractions on state-of-the-art database engines, such as Hyper [16], can be much faster than using the de-facto `numpy`-array implementation.

Chapter 13

Tensorgrad

Throughout this book we have manipulated tensor diagrams by hand. The *tensorgrad* library (`https://github.com/thomasahle/tensorgrad`) does the same manipulations symbolically in Python: tensors are represented as graphs with *named* edges, so there is no index bookkeeping, no transpose-juggling, and no einsum strings to keep consistent. On top of this representation the library implements symbolic differentiation, diagram simplification, and compilation to numeric PyTorch code. In this chapter we give a short overview of the design, and of the problems a tensor-diagram engine has to solve that a classical computer-algebra system does not.

13.1 Representation

Every expression is a subclass of `Tensor`. The leaves are `Variable` (a named tensor whose dimensions are sympy symbols) and the constants `Delta` (the copy/identity tensor, which also doubles as a hyper-edge of any order) and `Zero`. Composite expressions are built from `Sum`, `Product`, `Function`, and `Derivative`; a lightweight `Rename` node relabels edges lazily. Nonlinear functions such as `exp`, `log`, and `softmax` are described by a `FunctionSignature`, which declares which input edges are consumed, which output edges are produced, and how to differentiate—so users can add new functions without touching the core.

The key decision is that edges are identified by *name*, not by position. A matrix is a tensor with two edges, say i and j, and contraction happens simply by forming a `Product` of tensors that share an edge name. There is no distinction between a matrix and its transpose beyond which names its edges carry, which matches how tensor diagrams themselves work. The price is that renaming becomes a first-class operation the engine must handle carefully—for instance when a derivative introduces new edges that would otherwise clash with existing names.

13.2 Canonicalization

The central algorithmic problem is deciding when two diagrams are *the same*. Since a diagram is a labeled multigraph, this is exactly graph isomorphism: each expression can

produce a NetworkX graph of itself (`structural_graph`), free edges are turned into labeled nodes so they can participate in the matching, and two tensors are equal iff their graphs are isomorphic (`is_isomorphic`). For speed, tensorgrad first compares Weisfeiler–Lehman hashes (and, when the compiler backend is available, a refinement-based canonical fingerprint that is computed compositionally and memoized), and only falls back to an exact isomorphism search when the hashes collide.

Names make this subtler than plain graph isomorphism. Inside a `Sum`, the terms A and A^T are isomorphic *as graphs*, but $A + A^T$ must not simplify to $2A$: summands may only be merged under an isomorphism that respects the sum's edge names. Conversely, if the user declares A symmetric via `Variable.with_symmetries`, the symmetry becomes part of the graph, the renamed copy really is the same tensor, and $A + A^T = 2A$ follows automatically. Isomorphism testing also powers `Product` simplification, where equal subgraphs of a contraction are recognized and combined into powers (Figure 13.1), and common-subexpression elimination during evaluation, where a cached value for A can be reused for A^T by renaming its edges.

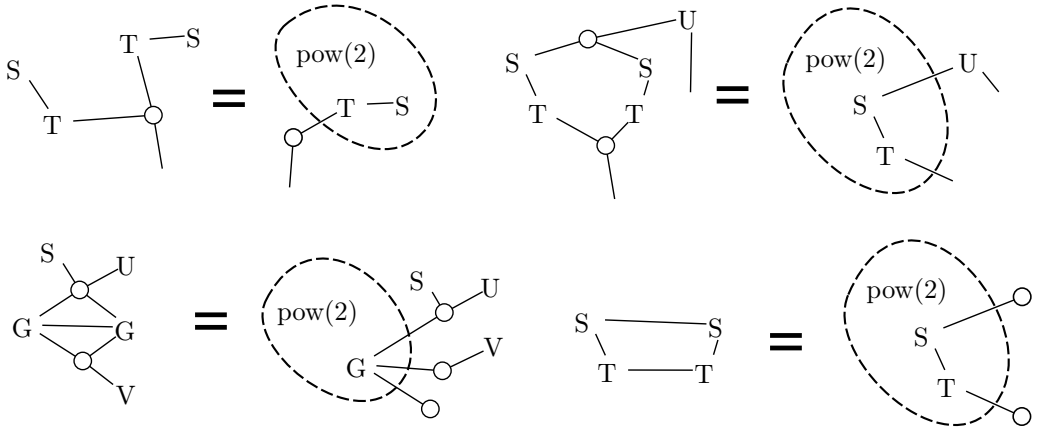


Figure 13.1: Combining equal parts of a product

13.3 Differentiation

Derivatives follow Penrose's convention from the derivative chapters: taking $\partial T/\partial X$ adds one new free edge to the diagram for every edge of X. Calling `t.grad(X)` wraps the expression in a `Derivative` node and pushes it one step inward using the rules of this book, diagrams-as-data: the derivative of a `Variable` with respect to itself becomes a product of `Delta` tensors connecting old edges to new ones; the derivative of a `Product` distributes by the product rule; a `Function` consults its `FunctionSignature` for the chain rule from Chapter 4. Because a derivative is just another node in the graph, higher-order derivatives like Hessians are simply `t.grad(x).grad(x)`, and `Expectation` (over Gaussian variables, using the identities from the statistics chapter) composes with everything else.

13.4 Simplification and Evaluation

Simplification is a set of local rewrite rules—evaluating deltas, flattening nested sums and products, cancelling zeros, merging isomorphic terms—applied in a depth-first pass by `simplify()`; `full_simplify()` repeats the pass until a fixed point is reached. The result can be rendered back as a diagram with `to_tikz` (that is how many figures in this book were produced), evaluated numerically with `evaluate`, which maps variables to PyTorch tensors and contracts products with `einsum`, or compiled to a standalone PyTorch function with `compile_to_callable` from `tensorgrad.extras.to_pytorch`.

As a small end-to-end example, here is the gradient of the least-squares loss from the machine-learning chapter:

```
import tensorgrad as tg
import tensorgrad.functions as F

b, x, y = tg.symbols("b x y")
X = tg.Variable("X", b, x)
Y = tg.Variable("Y", b, y)
W = tg.Variable("W", x, y)

error = X @ W - Y            # contracts the shared edge x
loss = F.frobenius2(error)   # contract all edges with themselves
grad = loss.grad(W).full_simplify()
```

The resulting diagram is $2X^T(XW - Y)$, with no matrix-shaped reasoning required from the user. For installation instructions, an interactive playground, and many more examples, see `https://github.com/thomasahle/tensorgrad`.

Bibliography

[1] Thomas D. Ahle, Michael Kapralov, Jakob B. T. Knudsen, Rasmus Pagh, Ameya Velingker, David P. Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 141–160, 2020.

[2] John Aldrich. *The Origins of Mathematical Words: A Comprehensive Dictionary of Latin, Greek, and Arabic Roots*. Mathematical Association of America, 2010.

[3] Mark Blacher, Julien Klaus, Christoph Staudt, Sören Laue, Viktor Leis, and Joachim Giesen. Efficient and portable einstein summation in SQL. *Proc. ACM Manag. Data*, 1(2):121:1–121:19, 2023.

[4] Lam Chi-Chung, P Sadayappan, and Rephael Wenger. On optimizing a class of multi-dimensional loops with reduction for parallel execution. *Parallel Processing Letters*, 7(02):157–168, 1997.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[6] Predrag Cvitanović. *Group Theory: Birdtracks, Lie's, and Exceptional Groups*. Princeton University Press, 2008.

[7] Jeffrey M Dudek, Leonardo Duenas-Osorio, and Moshe Y Vardi. Efficient contraction of large tensor networks for weighted model counting through graph decompositions. *arXiv preprint arXiv:1908.04381*, 2019.

[8] Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In David Maxwell Chickering and Joseph Y. Halpern, editors, *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, Banff, Canada, July 7-11, 2004*, pages 201–208. AUAI Press, 2004.

[9] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, 2021.

[10] greg (https://math.stackexchange.com/users/357854/greg). Proving that $\mathrm{vec}(a\,\mathrm{diag}(b)\,c) = ((c^t \otimes 1_a) \odot (1_c \otimes a))\,b$. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/2993406 (version: 2018-11-11).

[11] Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *Journal of Experimental Algorithmics (JEA)*, 23:1–34, 2018.

[12] William Rowan Hamilton. *Lectures on Quaternions*. Hodges and Smith, 1853.

[13] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. The alternating linear scheme for tensor optimization in the tensor train format. *SIAM Journal on Scientific Computing*, 34(2):A683–A713, 2012.

[14] T. C. Hu and M. T. Shing. Computation of matrix chain products. part I. *SIAM J. Comput.*, 11(2):362–373, 1982.

[15] T. C. Hu and M. T. Shing. Computation of matrix chain products. part II. *SIAM J. Comput.*, 13(2):228–251, 1984.

[16] Alfons Kemper and Thomas Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *2011 IEEE 27th International Conference on Data Engineering*, pages 195–206. IEEE, 2011.

[17] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. *SIAM Journal on Computing*, (0):FOCS21–174, 2023.

[18] Igor L Markov and Yaoyun Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, 2008.

[19] Eli Meirom, Haggai Maron, Shie Mannor, and Gal Chechik. Optimizing tensor network contraction using reinforcement learning. In *International Conference on Machine Learning*, pages 15278–15292. PMLR, 2022.

[20] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, 2015.

[21] Ivan V. Oseledets. Approximation of $2^d \times 2^d$ matrices using tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2130–2145, 2010.

[22] Ivan V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[23] Roger Penrose. Applications of negative dimensional tensors. In D. J. A. Welsh, editor, *Combinatorial Mathematics and its Applications*, pages 221–244. Academic Press, 1971.

[24] Robert N. C. Pfeifer, Jutho Haegeman, and Frank Verstraete. Faster identification of optimal contraction sequences for tensor networks. *Phys. Rev. E*, 90:033315, 2014.

[25] Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 239–247, 2013.

[26] Beheshteh T. Rakhshan and Guillaume Rabusseau. Tensor cookbook: Mastering tensors through diagrams, 2026. arXiv:2605.16610.

[27] Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. K-way hypergraph partitioning via n-level recursive bisection. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 53–67. SIAM, 2016.

[28] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, 2011.

[29] Roman Schutski, Danil Lykov, and Ivan Oseledets. Adaptive algorithm for quantum circuit simulation. *Phys. Rev. A*, 101:042335, Apr 2020.

[30] Daniel G. A. Smith and Johnnie Gray. opt_einsum - a Python package for optimizing contraction order for einsum-like expressions. *Journal of Open Source Software*, 3(26):753, 2018.

[31] Christoph Staudt, Mark Blacher, Julien Klaus, Farin Lippmann, and Joachim Giesen. Improved cut strategy for tensor network contraction orders. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms, SEA 2024, July 23-26, 2024, Vienna, Austria*, volume 301 of *LIPIcs*, pages 27:1–27:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[32] Mihail Stoian and Andreas Kipf. Dpconv: Super-polynomially faster join ordering, 2024.

[33] Mihail Stoian, Richard M. Milbradt, and Christian B. Mendl. On the optimal linear contraction order of tree tensor networks, and beyond. *SIAM Journal on Scientific Computing*, 46(5):B647–B668, 2024.

[34] Ben Strasser. Computing tree decompositions with flowcutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017.

[35] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical Review Letters*, 91(14):147902, 2003.

[36] Jianyu Xu, Ling Liang, Lei Deng, Changyun Wen, Yuan Xie, and Guoqi Li. Towards a polynomial algorithm for optimal contraction sequence of tensor networks from trees. *Phys. Rev. E*, 100:043309, 2019.

[37] F. Frances Yao. Efficient dynamic programming using quadrangle inequalities. In Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, editors, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 429–435. ACM, 1980.

Chapter 14

Appendix

Contains some proofs, such as of equation 524 or 571. They are pretty long and could be useful for contrasting with the diagram proofs.