

The Tensor Cookbook

Thomas Dybdahl Ahle

February 9, 2025

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 amendable 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 make the 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. [1, 10].

Contents

1	Introduction	1
1.1	Tensor Diagrams	7
1.2	The Copy Tensor	8
1.3	Sums of Tensors	9
1.4	Transposition	10
1.4.1	Higher order	11
1.5	Trace	11
1.5.1	Higher order traces	11
1.6	Eigenvalues	12
1.7	Symmetry and Symmetrization	12
1.8	Covariance and Contravariance	13
1.9	Exercises	13
2	Simple Derivatives	14
2.1	Derivatives of Matrices, Vectors and Scalar Forms	14
2.1.1	First Order	14
2.1.2	Second Order	15
2.1.3	Higher Order	16
2.2	Derivatives of Traces	16
2.2.1	First Order	16
2.2.2	Second Order	17
2.2.3	Higher Order	18
2.3	Exercises	18
3	Kronecker and Vec Operator	20
3.1	Flattening	20
3.2	The Kronecker Product	21
3.3	The Vec Operator	22
3.4	Kronecker Vector Product	23
3.5	General Matrification	23
3.5.1	The Lyapunov Equation	23
3.5.2	Encapsulating Sum	24
3.6	The Hadamard Product	24
3.7	Khatri–Rao product	25
3.8	Tracy-Singh product	25

3.8.1	Stacking	25
3.9	Derivatives	26
3.10	Exercises	26
4	Functions	28
4.1	The Chain Rule	29
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	31
4.2.1	Known derivatives	31
4.2.2	Pseudo-linear forms	32
4.2.3	Trace identity	32
4.2.4	Taylor	33
4.3	Exercises	33
5	Statistics and Probability	37
5.1	Definition of Moments	37
5.1.1	Expectation of Linear Combinations	37
5.1.2	Linear Forms	38
5.1.3	Quadratic Forms	38
5.1.4	Cubic Forms	39
5.2	Cumulants	40
5.2.1	Quartic Forms	41
5.3	Weighted Scalar Variable	41
5.4	Gaussian Moments	41
5.4.1	Gaussian Integration by Parts	42
5.4.2	Cubic forms	42
5.4.3	Mean of Quartic Forms	42
5.4.4	Mixture of Gaussians	43
5.4.5	Derivatives	44
5.5	Exercises	44
6	Determinant and Inverses	45
6.1	Determinant	45
6.2	Inverses	46
7	Advanced Derivatives	47
7.1	Derivatives of vector norms	47
7.1.1	Two-norm	47
7.2	Derivatives of matrix norms	48
7.3	Derivatives of Structured Matrices	48
7.3.1	Symmetric	48
7.3.2	Diagonal	48
7.3.3	Toeplitz	48
7.4	Derivatives of a Determinant	48
7.5	General forms	48
7.6	Linear forms	48

7.7	Square forms	48
7.8	From Stack Exchange	48
7.9	Derivatives of an Inverse	48
7.9.1	Trace Identities	48
7.10	Derivatives of Eigenvalues	49
7.11	Exercises	49
8	Special Matrices	50
8.0.1	Block matrices	50
8.0.2	The Discrete Fourier Transform Matrix	50
8.0.3	Fast Kronecker Multiplication	50
8.0.4	Hermitian Matrices and skew-Hermitian	54
8.0.5	Idempotent Matrices	54
8.0.6	Orthogonal matrices	54
8.0.7	Positive Definite and Semi-definite Matrices	54
8.0.8	Singleentry Matrix, The	54
8.0.9	Symmetric, Skew-symmetric/Antisymmetric	54
8.0.10	Toeplitz Matrices	54
8.0.11	Units, Permutation and Shift	54
8.0.12	Vandermonde Matrices	54
9	Decompositions	55
9.1	Higher-order singular value decomposition	55
9.2	Rank Decomposition	55
9.2.1	Border Rank	55
9.3	Fast Matrix Multiplication	55
10	Machine Learning Applications	58
10.1	Least Squares	58
10.2	Hessian of Cross Entropy Loss	58
10.3	Convolutional Neural Networks	58
10.4	Transformers / Attention	58
10.5	Tensor Sketch	58
10.6	Reinforcement Learning	58
11	Tensor Algorithms	59
11.1	Tensor Contraction Orders	59
11.1.1	Algorithms	60
12	Tensorgrad	61
12.1	Isomorphisms	61
12.1.1	In Products	61
12.1.2	In Sums	62
12.1.3	In Evaluation	63
12.1.4	In Variables	63
12.1.5	In Constants	63
12.1.6	In Functions	64
12.1.7	In Derivatives	64

<i>CONTENTS</i>	6
12.1.8 Other	64
12.2 Renaming	64
12.3 Evaluation	64
12.3.1 Products	65
12.4 Simplification Rules	66
13 Appendix	69

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 follow table shows how some basic operations can be written with tensor diagrams:

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

We think of vectors and matrices as tensors of order 1 and 2. The order corresponds to the number of dimensions in their $[\dots]$ 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 or 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 $i = 1 \dots n$. 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:

$$\begin{array}{c} | \\ \diagup T \diagdown \\ x \end{array} \quad \text{and} \quad \begin{array}{c} | \\ x \diagup T \diagdown \end{array} \quad \text{and} \quad \begin{array}{c} x \\ \diagup T \diagdown \end{array}$$

To be perfectly precise about what each one means, we should give the edges labels. For example we would write $\begin{array}{c} | \\ \diagup T \diagdown \\ x \end{array}$ 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.

$$Y_{i,j} = \sum_{k,l,m,n,o} A_{i,k} B_{l,n,o} C_{j,k,l,m} D_{m,n} E_o \Leftrightarrow \begin{array}{c} i \\ \diagdown \\ Y \end{array} = \begin{array}{c} i \\ \diagdown \\ A \\ k \\ \diagup \\ C \\ j \end{array} \begin{array}{c} l \\ \diagup \\ B \\ n \\ \diagdown \\ D \\ m \end{array} \begin{array}{c} o \\ \diagdown \\ E \end{array}$$

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 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 11.1 for algorithms to find the best contraction order, and approximate contraction methods.

Note that tensor graphs are not always connected. We already saw that the outer product of two vectors can be written $-a \ b-$. 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”, “kro-necker 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:

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

Or, using Iversonian notation,¹ $\circ_{i,j,k,\dots} = [i = j = k = \dots]$. 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

$$D_a = \text{diag}_a$$

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

$$a \circ b = \text{diag}_a \text{diag}_b$$

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”:

$$D_a D_b = \text{diag}_a \text{diag}_b = \text{diag}_a \text{diag}_b = \text{diag}_{a \circ b} = D_{a \circ b}.$$

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

¹For 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 diagram shows an equality between three expressions. The first expression is a triangle with vertices represented by small circles. The left vertex is connected to a label 'S', and the right vertex is connected to a label 'T'. A dashed line encloses the top two vertices and the edge between them. The second expression is a V-shaped graph with a single vertex at the bottom connected to two vertices above it, labeled 'S' and 'T' respectively. The third expression is a horizontal line segment connecting 'S' and 'T'.

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 $J = \begin{smallmatrix} \circ & \circ \\ \text{---} & \text{---} \\ \circ & \circ \end{smallmatrix}$, 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, $\begin{smallmatrix} \times & \times \\ \text{---} & \text{---} \\ \times & \times \end{smallmatrix}$.

1.3 Sums of Tensors

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

To handle this important operation, Penrose suggesting 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:

$$\begin{array}{c} \text{R} \\ \text{U} \end{array} \begin{array}{c} i \\ k \\ j \end{array} \left(\begin{array}{c} \text{T} \\ \text{M} \end{array} \begin{array}{c} i \\ j \\ k \end{array} + \begin{array}{c} k \\ j \end{array} \text{V} \right) = \begin{array}{c} \text{T} \\ \text{M} \\ \text{U}-\text{R} \end{array} + \begin{array}{c} \text{R} \\ \text{U} \\ \text{V} \end{array}$$

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 $\frac{i}{j} \circ x$ is an outer product between x and the all one 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 $\frac{i}{j} \circ x$.

Note that we typically don’t talk about “rows” or “columns” when dealing with tensors, but simply use the name edge (sometimes axis) 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 on "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:

$$(-A-)^T = \overline{\underbrace{A}} = -A^T-.$$

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:

$$({}^iA-^j)^T = {}^{i-j}A^{i-j}$$

Renaming can be done using multiplication by identity matrices:

$$({}^iA-^j)(_{j-o-k}) = {}^iA^{j-k},$$

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

$$[({}^iA-^j)(_{j-o-k})]({}_{k-o-j}) = {}^iA-^j \neq ({}^iA-^j)[({}_{j-o-k})(_{k-o-j})] = {}^iA-^j \circlearrowleft,$$

where \circlearrowleft 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:

$$\begin{aligned} (A^T)^T &= A & \overline{\underbrace{A}} &= -A- \\ (A+B)^T &= A^T + B^T & \overline{(-A- + -B-)} &= \overline{-A-} + \overline{-B-} \end{aligned} \quad (4)$$


$$(AB)^T = B^T A^T \quad \overline{A-B} = \overline{\underbrace{V}}_{-B} = -B-V- \quad (5)$$

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 $\text{Tr}(A) = \sum_i A_{i,i}$. 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:

$$\sum_{i=1}^n A_{ii} = \text{Tr}(A) = \text{Tr}(AI) \quad \text{A} = \text{A} \circ \text{I} \quad (11)$$

$$\text{Tr}(A) = \text{Tr}(A^T) \quad \text{A} = \text{V} \quad (13)$$

$$\text{Tr}(AB) = \text{Tr}(BA) \quad \text{A} \text{---} \text{B} = \text{B} \text{---} \text{A} = \text{B} \text{---} \text{A} \quad (14)$$

$$\text{Tr}(A + B) = \text{Tr}(A) + \text{Tr}(B) \quad \text{A} + \text{B} = \text{A} + \text{B} \quad (15)$$

$$\begin{aligned} \text{Tr}(ABC) &= \text{Tr}(BCA) & \text{A} \text{---} \text{B} \text{---} \text{C} &= \text{B} \text{---} \text{C} \text{---} \text{A} \\ &= \text{Tr}(CAB) & &= \text{C} \text{---} \text{A} \text{---} \text{B} \end{aligned} \quad (16)$$

$$\begin{aligned} a^T a &= \text{Tr}(aa^T) & a \text{---} a &= \text{Tr}(-a \text{---} a) \\ & & &= \text{a} \end{aligned} \quad (17)$$

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 A of order n , the trace is just the product with the order n copy tensor:

$$\text{Tr}(\text{I}^{\text{I}}) = \text{I} \circ \text{I}.$$

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

$$\text{Tr}_{i,j}(\text{I}^{\text{I}}) = \text{I}.$$

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:

$$-A- = -Q-\underset{\lambda}{\circ}-Q^{-1}- \quad (1.1)$$

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

$$\text{Tr}(A) = \text{A} = \text{Q}-\underset{\lambda}{\circ}-\text{Q}^{-1} = \underset{\lambda}{\circ} = \underset{\lambda}{\circ} = \sum_i \lambda_i \quad (12)$$

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 summing over all permutations of its indices. We write this using a squiggly line over the symmetrized edges:

$$\text{sym} = \text{=} + \text{X}, \quad \text{sym} = \text{=} + \text{X} + \text{X} + \text{X} + \text{X} + \text{X}$$

For example, if A is a square matrix, $A + A^T = \text{sym} A \text{sym} = -A- + \text{sym} A \text{sym}$. There is a complementary notion of *anti-symmetrization* (or *skew-symmetrization*), where we sum over *all permutations with appropriate sign*. For instance, a rank-2 skew-symmetric matrix A satisfies $A_{ij} = -A_{ji}$. We can anti-symmetrize using a flat thick line: $\text{anti-sym} A \text{anti-sym} = -A- - \text{sym} A \text{sym}$. In higher-rank cases, the sign of each term is determined by the parity of the permutation.

Both tensors are idempotent, since symmetrizing a symmetric tensor has no effect:

$$\text{sym} \text{sym} = \text{sym}, \quad \text{anti-sym} \text{anti-sym} = \text{anti-sym}.$$

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 generalize to the concepts of covariance and contravariance. With notion, we keep track of “input” and “output” vectors. We'll also need to distinguish between things such as the identity matrix, $-o-$, \succ , and \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 $A_1, A_2, \dots, A_n \in \mathbb{R}^{n \times n}$, and vectors $v_1, v_2, \dots, v_n \in \mathbb{R}^n$, draw, using tensor diagrams, the matrix made of vectors $A_1 v_1, A_2 v_2, \dots, A_n v_n$.

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

$$\frac{dT}{dU} = \text{Diagram: A circle with a 'T' inside. Two lines enter from the left, and two lines exit from the top-right. A black dot is on the top-right line, with two more lines extending from it, representing a rank-5 tensor.$$

This notation follows Penrose. The two extra lines coming from the black dot on the circle makes 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:

$$\succ(T)\text{---}$$

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

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.

$$\frac{\partial x^T a}{\partial x} = a \qquad (x \text{---} \text{dot}) = (x) \text{---} a = -a \qquad (69)$$

$$\frac{\partial a^T x}{\partial x} = a \qquad (a \text{---} \text{dot}) = a \text{---} (x) = a \text{---} \text{hook} = -a \qquad (69)$$

$$\frac{\partial a^T X b}{\partial X} = ab^T \qquad (a \text{---} X \text{---} b) \text{---} \text{dot} = a \text{---} (X) \text{---} b = a \text{---} \text{hook} \text{---} b \qquad (70)$$

$$\frac{\partial X}{\partial X_{i,j}} = J^{i,j} \quad (-X - \overset{i}{\underset{j}{\bullet}} = \underset{i}{\frown} \underset{j}{\smile} \quad (73)$$

$$\begin{aligned} \frac{\partial (XA)_{i,j}}{\partial X_{m,n}} &= (J^{m,n} A)_{i,j} & (\overset{i}{\underset{j}{\bullet}} X - A - \overset{m}{\underset{n}{\bullet}} &= -(X) \overset{m}{\underset{n}{\bullet}} A - \\ & & &= \overset{i}{\underset{n}{\frown}} \underset{j}{\smile} A - \end{aligned} \quad (74)$$

2.1.2 Second Order

The second order derivatives are follow from the product rule:

$$\begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \text{---} T \text{---} U \text{---} \end{array} = \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \text{---} T \text{---} U \text{---} \end{array} + \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \text{---} T \text{---} U \text{---} \end{array}$$

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

$$\begin{aligned} \frac{\partial}{\partial X_{i,j}} \sum_{k,l,m,n} X_{k,l} X_{m,n} &= \left(\sum_{k,l} X_{k,l} \right)^2 & \left(\overset{i}{\underset{j}{\bullet}} X - \overset{i}{\underset{j}{\bullet}} \right) &= \overset{i}{\underset{j}{\bullet}} X - \overset{i}{\underset{j}{\bullet}} + \overset{i}{\underset{j}{\bullet}} (X) - \overset{i}{\underset{j}{\bullet}} \\ &= 2 \sum_{k,l} X_{k,l} & &= 2 \overset{i}{\underset{j}{\bullet}} X - \overset{i}{\underset{j}{\bullet}} \end{aligned} \quad (76)$$

$$\begin{aligned} \frac{\partial b^T X^T X c}{\partial X} &= X(b c^T + c b^T) & (b - X^T - X - c) \overset{i}{\underset{j}{\bullet}} &= b - X^T - (X) \overset{i}{\underset{j}{\bullet}} - c \\ & & &+ b - (X^T) \overset{i}{\underset{j}{\bullet}} - X - c \\ & & &= b - X^T - \overset{i}{\underset{j}{\bullet}} - c \\ & & &+ b - \overset{i}{\underset{j}{\bullet}} - X - c \\ & & &= -X - (-b c' + -c b') \end{aligned} \quad (77)$$

$$\begin{aligned} \frac{\partial}{\partial X_{i,j}} (X^T B X)_{k,l} &= \delta_{l,j} (X^T B)_{k,i} & (\overset{i}{\underset{j}{\bullet}} X^T - B - X - \overset{i}{\underset{j}{\bullet}}) &= \overset{i}{\underset{j}{\bullet}} X^T - B - \overset{i}{\underset{j}{\bullet}} \\ &+ \delta_{k,j} (B X)_{i,l} & &+ \overset{i}{\underset{j}{\bullet}} B - X - \overset{i}{\underset{j}{\bullet}} \end{aligned} \quad (79)$$

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

$$\begin{aligned} \frac{\partial}{\partial x} x^T B x &= (B + B^T) x & (x - B - x) \overset{i}{\underset{j}{\bullet}} &= -B - x + x - B \\ & & &= (-B - + -B^T -) - x \\ & & &= x - \overset{i}{\underset{j}{\bullet}} \left(\overset{i}{\underset{j}{\bullet}} B - \overset{i}{\underset{j}{\bullet}} \right) \end{aligned} \quad (81)$$

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

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.

$$\begin{aligned} \frac{\partial}{\partial \mathbf{X}} \mathbf{a}^T \mathbf{X}^n \mathbf{b} &= \sum_{r=0}^{n-1} (\mathbf{X}^r)^T \mathbf{a} \mathbf{b}^T (\mathbf{X}^{n-1-r})^T & (a - X - \dots - X - b) \nearrow & \quad (91) \\ &= \sum_{r=0}^{n-1} a - X^r \nearrow X^{n-r-1} - b \\ &= \sum_{r=0}^{n-1} -(X^r)^T - a \quad b - (X^{n-r-1})^T - \end{aligned}$$

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.

$$\begin{aligned} \frac{\partial}{\partial X} \text{Tr}(XA) &= A^T \\ \left(\overbrace{X-A} \right) \bullet &= \overbrace{(X \bullet - A)} \\ &= \overbrace{\succ \prec A} \\ &= -A^T \end{aligned} \quad (100)$$

$$\begin{aligned}
\frac{\partial}{\partial X} \text{Tr}(AXB) &= A^T B^T & (\overbrace{A-X-B}^{\nearrow}) &= \overbrace{A-(X)-B}^{\nearrow} \\
& & &= \overbrace{A \rightarrow \leftarrow B}^{\nearrow} \\
& & &= -A^T - B^T -
\end{aligned} \tag{101}$$

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

2.2.2 Second Order

$$\begin{aligned}
\frac{\partial}{\partial X} \text{Tr}(X^2) &= 2X^T & (\overbrace{X-X}^{\nearrow}) & \\
& & &= \overbrace{(X)-X}^{\nearrow} + \overbrace{X-(X)}^{\nearrow} \\
& & &= \overbrace{\nearrow \leftarrow X}^{\nearrow} + \overbrace{X \rightarrow \searrow}^{\nearrow} \\
& & &= 2 - X^T -
\end{aligned} \tag{106}$$

$$\begin{aligned}
\frac{\partial}{\partial X} \text{Tr}(X^2 B) &= (XB + BX)^T & (\overbrace{X-X-B}^{\nearrow}) & \\
& & &= \overbrace{(X)-X-B}^{\nearrow} + \overbrace{X-(X)-B}^{\nearrow} \\
& & &= \overbrace{\nearrow \leftarrow X-B}^{\nearrow} + \overbrace{X \rightarrow \leftarrow B}^{\nearrow} \\
& & &= -B^T - X^T - + -X^T - B^T -
\end{aligned} \tag{107}$$

$$\begin{aligned}
\frac{\partial}{\partial X} \text{Tr}(X^T B X) &= \frac{\partial}{\partial X} \text{Tr}(X X^T B) & (\overbrace{X^T-B-X}^{\nearrow}) & \\
&= \frac{\partial}{\partial X} \text{Tr}(B X X^T) & &= \overbrace{(X^T)-B-X}^{\nearrow} + \overbrace{X^T-B-(X)}^{\nearrow} \\
&= (B + B^T)X & &= \overbrace{\leftarrow B-X}^{\nearrow} + \overbrace{X^T-B \rightarrow \searrow}^{\nearrow} \\
& & &= -B-X- + -B^T-X-
\end{aligned} \tag{108, 109, 110}$$

$$\begin{aligned}
\frac{\partial}{\partial X} \text{Tr}(X B X^T) &= \frac{\partial}{\partial X} \text{Tr}(X^T X B) & (\overbrace{X-B-X^T}^{\nearrow}) & \\
&= \frac{\partial}{\partial X} \text{Tr}(B X^T X) & &= \overbrace{(X)-B-X^T}^{\nearrow} + \overbrace{X-B-(X^T)}^{\nearrow} \\
&= X(B^T + B) & &= \overbrace{\nearrow \leftarrow B-X^T}^{\nearrow} + \overbrace{X-B-}^{\nearrow} \\
& & &= -X-B^T- + -X-B-
\end{aligned} \tag{111, 112, 113}$$

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

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

However that is clearly not that case. Such substitution would only work for a linear function, not a quadratic. In general it is the case that $\frac{\partial}{\partial X} f(X)^T \neq \frac{\partial}{\partial X} f(X^T)$.

2.2.3 Higher Order

$$\begin{aligned} \frac{\partial}{\partial X} \text{Tr}(X^n) &= n(X^{n-1})^T && \overbrace{(X-X-X-\dots-X)}^{\text{diagram}} \quad (121) \\ &= \sum_{r=0}^{n-1} \overbrace{X^r \rightarrow X^{n-r-1}}^{\text{diagram}} \\ &= n(X^T)^{n-1} \end{aligned}$$

$$\begin{aligned} \frac{\partial}{\partial X} \text{Tr}(AX^n) &= \sum_{r=0}^{n-1} (X^r AX^{n-1-r})^T && \overbrace{(A-X-X-\dots-X)}^{\text{diagram}} \quad (122) \\ &= \sum_{r=0}^{n-1} \overbrace{A-X^r \rightarrow X^{n-r-1}}^{\text{diagram}} \\ &= \sum_{r=0}^{n-1} -(X^r)^T - A^T - (X^{n-r-1})^T - \end{aligned}$$

2.3 Exercises

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

Exercise 5. Find the second derivative of

$$x^T A^T x x^T A x$$

with respect to x .

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

Exercise 7. Show the derivatives:

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

$$\frac{\partial}{\partial X} b^T X^T DX c = D^T X b c^T + DX c b^T \quad (82)$$

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

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

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

$$\frac{\partial}{\partial X} \text{Tr}(X^T X) = \frac{\partial}{\partial X} \text{Tr}(XX^T) = 2X \quad (115)$$

$$\frac{\partial}{\partial X} \text{Tr}(B^T X^T CXB) = C^T XBB^T + CXBB^T \quad (116)$$

$$\frac{\partial}{\partial X} \text{Tr}[X^T BXC] = BXC + B^T XC^T \quad (117)$$

$$\frac{\partial}{\partial X} \text{Tr}(AXBX^T C) = A^T C^T XB^T + CAXB \quad (118)$$

$$\frac{\partial}{\partial X} \text{Tr}[(AXB + C)(AXB + C)^T] = 2A^T(AXB + C)B^T \quad (119)$$

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

$$\begin{aligned} \frac{\partial}{\partial X} \text{Tr}[B^T X^T CXX^T CXB] &= CXX^T CXBB^T \\ &\quad + C^T XBB^T X^T C^T X \\ &\quad + CXBB^T X^T CX \\ &\quad + C^T XX^T C^T XBB^T \end{aligned} \quad (123)$$

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 cludge 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:

$$\triangleright_{i,j,k} = \begin{array}{c} i \\ \diagdown \\ \diagup \\ j \end{array} \triangleright \begin{array}{c} k \\ \diagup \\ \diagdown \end{array} = [i + jn = k].$$

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.

Using this notation we can write

$$\text{vec}(X)_k = \sum_{i,j} \triangleright_{i,j,k} X_{i,j} = X \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} k \\ \diagup \\ \diagdown \end{array}.$$

Some basic properties of \triangleright :

$$\begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} = \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \quad (3.1)$$

$$\begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} = \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \quad (3.2)$$

$$\begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} = \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \quad (3.3)$$

$$\begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} = \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \begin{array}{c} \text{---} \triangleright \end{array} \quad (3.4)$$

3.2 The Kronecker Product

The Kronecker product of an $m \times n$ matrix A and an $r \times q$ matrix B , is an $mr \times nq$ 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 $(A \otimes B)_{p(r-1)+v, q(s-1)+w} = A_{rs}B_{vw}$, 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": $A \otimes B = \begin{matrix} \text{A} \\ \text{B} \end{matrix}$.

The Kronecker product has the following properties:

$$A \otimes (B + C) = A \otimes B + A \otimes C \quad \begin{matrix} \text{A} \\ \text{(B+C)} \end{matrix} = \begin{matrix} \text{A} \\ \text{B} \end{matrix} + \begin{matrix} \text{A} \\ \text{C} \end{matrix} \quad (506)$$

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \quad \begin{matrix} \text{A} \\ \text{B} \\ \text{C} \end{matrix} = \begin{matrix} \text{A} \\ \text{B} \\ \text{C} \end{matrix} = \begin{matrix} \text{A} \\ \text{B} \\ \text{C} \end{matrix} \quad (508)$$

$$aA \otimes bB = ab(A \otimes B) \quad \begin{matrix} a \text{ A} \\ b \text{ B} \end{matrix} = \begin{matrix} ab \\ \text{A} \\ \text{B} \end{matrix} \quad (509)$$

$$(A \otimes B)^T = A^T \otimes B^T \quad \begin{matrix} \text{A} \\ \text{B} \end{matrix} = \begin{matrix} \text{A} \\ \text{B} \end{matrix} \quad (510)$$

$$(A \otimes B)(C \otimes D) = AC \otimes BD \quad \begin{matrix} \text{A} \\ \text{B} \end{matrix} \begin{matrix} \text{C} \\ \text{D} \end{matrix} = \begin{matrix} \text{A-C} \\ \text{B-D} \end{matrix} \quad (511)$$

$$(A \otimes I)(I \otimes B) = A \otimes B \quad \begin{matrix} \text{A} \\ \text{B} \end{matrix} = \begin{matrix} \text{A} \\ \text{B} \end{matrix} \quad (511b)$$

$$\text{Tr}(A \otimes B) = \text{Tr}(A)\text{Tr}(B) \quad \begin{matrix} \text{A} \\ \text{B} \end{matrix} = \begin{matrix} \text{A} \\ \text{B} \end{matrix} = \text{A} \text{ B} \quad (515)$$

$$\begin{aligned} \text{eig}(A \otimes B) &= \text{eig}(A)\text{eig}(B) \\ &= \text{Diagram 1} \\ &= \text{Diagram 2} \\ &= \text{Diagram 3} \end{aligned} \tag{519}$$

This is easier to see when we consider that $V = \frac{\circ}{\text{M}} \overline{\nearrow} \text{---}$ 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 $\text{diag}(\text{vec}(M))$.

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

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

$$\text{vec}(A^T X B) = \text{vec}(X)^T (A \otimes B) \quad \text{X} \begin{array}{c} \diagup A \\ \diagdown B \end{array} \text{ } = \text{X} \begin{array}{c} \diagup A \\ \diagdown B \end{array} \text{ } \quad (520)$$

$$\text{Tr}(A^T B) = \text{vec}(A)^T \text{vec}(B) \qquad A \bigcirc B = A \bowtie B \qquad (521)$$

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

$$\text{vec}(aA) = a \text{vec}(A) \qquad aA \rhd = a A \rhd \qquad (523)$$

$$a^T X B X^T c = \text{vec}(X)^T (B \otimes c a^T) \text{vec}(X) \quad a-X-B-X-c = \text{X} \begin{array}{c} \text{B} \\ \swarrow \quad \searrow \\ a \quad b \end{array} \text{X} \quad (524)$$

$$= X \frown \begin{array}{c} \text{B} \\ a \quad b \end{array} \text{X}$$

3.4 Kronecker Vector Product

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

$$A \otimes v = \begin{array}{c} \text{A} \\ \diagdown \quad \diagup \\ \text{v} \end{array} \text{---} \quad (3.5)$$

$$A \otimes v^T = \text{---} \begin{array}{c} \text{A} \\ \diagup \quad \diagdown \\ \text{v} \end{array} \quad (3.6)$$

$$v \otimes A = \begin{array}{c} \text{v} \\ \diagdown \quad \diagup \\ \text{A} \end{array} \text{---} \quad (3.7)$$

$$v^T \otimes A = \text{---} \begin{array}{c} \text{v} \\ \diagup \quad \diagdown \\ \text{A} \end{array} \quad (3.8)$$

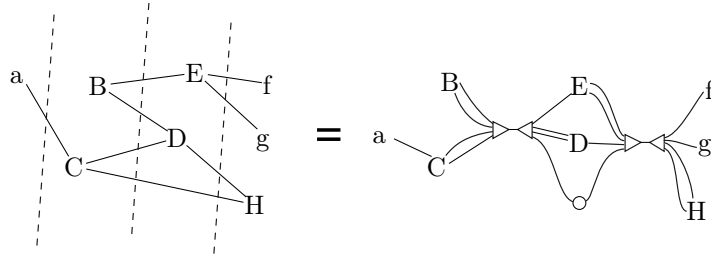
Similarly, we can define the Kronecker product between two vectors:

$$v \otimes w = \begin{array}{c} \text{v} \\ \diagdown \quad \diagup \\ \text{w} \end{array} \quad (3.9)$$

$$v^T \otimes w^T = \begin{array}{c} \text{v} \\ \diagup \quad \diagdown \\ \text{w} \end{array} \quad (3.10)$$

3.5 General Matrifaction

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, aM_1M_2b , 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

$$AX + XB = C. \quad (272)$$

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:

$$\left(\begin{array}{c} -A-X- \\ + \quad -X-B- \end{array} \right) \curvearrowright = X \overset{A}{\curvearrowright} + X \underset{B}{\curvearrowright} = X \curvearrowright \left(\begin{array}{c} \curvearrowleft A \curvearrowright \\ \curvearrowleft B \curvearrowright \end{array} \right) =$$

after which we can take the normal matrix inverse to get

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

3.5.2 Encapsulating Sum

This is a generalization of the previous equation.

$$\sum_n A_n X B_n = C \quad (274)$$

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

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:

$$A \circ B = - \circ \begin{array}{c} A \\ B \end{array} \circ -.$$

Some properties of the Hadamard product are:

$$\begin{aligned} x^T(A \circ B)y &= \text{tr}(A^T D_x B D_y) & x \text{---} \begin{array}{c} A \\ B \end{array} \text{---} y &= \overbrace{A^T \text{---} \underset{x}{\circ} \text{---} B \text{---} \underset{y}{\circ}} \\ (A \otimes B) \circ (C \otimes D) &= (A \circ C) \otimes (B \circ D) & \begin{array}{c} \text{---} \circ \begin{array}{c} A \\ B \\ C \\ D \end{array} \circ \text{---} \end{array} &= \begin{array}{c} \text{---} \begin{array}{c} A \\ B \\ C \\ D \end{array} \circ \text{---} \end{array} \end{aligned}$$

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.

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

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

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

$$A * B = \text{diag}_B(A) \circ -$$

$$A \bullet B = - \circ \text{diag}_B(A)$$

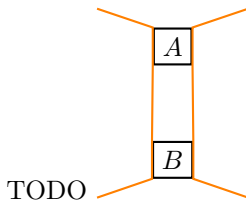
Clearly the two are identical up to transpose. Indeed, $(A * B)^T = B^T \bullet A^T$ and $(A \bullet B)^T = B^T * A^T$.

There are multiple “mixed product” identities:

$$(A \bullet B)(C \otimes D) = (AC) \bullet (BD) \quad - \circ \text{diag}_B(A) \text{diag}_D(C) = - \circ \text{diag}_{B-D}(A-C)$$

$$(Ax) \circ (By) = (A \bullet C)(x \otimes y) \quad - \circ \text{diag}_B(A-x) = - \circ \text{diag}_B(A) \text{diag}_y(x)$$

3.8 Tracy-Singh product



3.8.1 Stacking

Can be part of Kronecker section

From Josh: Proposition 2.5. For any field \mathbb{F} , integers d_1, d_2, d_3, d_4 and matrices $X_1 \in \mathbb{F}^{d_1 \times d_2}$, $X_2 \in \mathbb{F}^{d_2 \times d_3}$, $X_3 \in \mathbb{F}^{d_1 \times d_4}$, and $X_4 \in \mathbb{F}^{d_4 \times d_3}$, we have

$$X_1 \times X_2 + X_3 \times X_4 = (X_1 \mid X_3) \times \left(\frac{X_2}{X_4} \right),$$

where we are writing ‘|’ to denote matrix concatenation.

With tensor diagrams we can write stacking along a new axis i as

$$\text{stack}_i(X, Y) = \frac{e^{(0)-i}}{-X-} + \frac{e^{(1)-i}}{-Y-}$$

where $e_i^{(i)} = 1$ and 0 elsewhere.

From this we easily get the identity

$$(A \mid C) \left(\frac{B}{D} \right) = \text{stack}_i(A, C) \text{stack}_i(B, D) \quad (3.11)$$

$$= \left(\frac{e^{(0)-i}}{{}_k A_{-j}} + \frac{e^{(1)-i}}{{}_k C_{-j}} \right) \left(\frac{e^{(0)-i}}{{}_j B_{-m}} + \frac{e^{(1)-i}}{{}_j D_{-m}} \right) \quad (3.12)$$

$$= \frac{e^{(0)} - e^{(0)}}{k} A - B - \frac{e^{(1)} - e^{(1)}}{k} C - D - \frac{e^{(1)} - e^{(1)}}{m} \quad (3.13)$$

$$= AB + CD \quad (3.14)$$

TODO: Relation to direct sum, which is basically stacking + flattening. Or maybe it's nicer to do it by hadamard producting with the e_i vector. Also notice that this is what quantum comp people call “controlling”.

Also have a bunch of properties like $A \otimes (B \oplus C) = A \otimes B \oplus A \otimes C$.

3.9 Derivatives

3.10 Exercises

Exercise 10. Let $J = \text{Tr}[(I_N \otimes F)^T A (I_N \otimes F) B]$ 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 [8]:

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

Here $\text{diag}(b)$ is a diagonal matrix with the vector b on the diagonal, $\mathbf{1}$ 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 $\begin{smallmatrix} b \\ \text{---} \otimes \\ A \end{smallmatrix}$, 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 $\text{Tr}(A^T B) = 1^T \text{vec}(A \circ B)$, where 1 is a vector of the appropriate size.

Exercise 15. Find the derivative of

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

with respect to X .

Exercise 16.

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

Exercise 17. Take the derivative of

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

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

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

Exercise 18. Verify the following identity:

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

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

Exercise 19. 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 $\overset{m}{f} \leftarrow^n x$ when f is a function from \mathbb{R}^n to \mathbb{R}^m . If f is element-wise, we write $(f \leftarrow x) \overset{n}{\leftarrow} \mathbb{R}^n$. 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: $-f \leftarrow (-M - x)$. It may be helpful with some more examples:

$f : \mathbb{R}^n \rightarrow \mathbb{R}$	$f(x) \in \mathbb{R}$	$f \leftarrow^n x$	(Scalar function)
$g : \mathbb{R}^n \rightarrow \mathbb{R}^m$	$g(x) \in \mathbb{R}^m$	$\overset{m}{g} \leftarrow^n x$	(Vector function)
$h : \mathbb{R} \rightarrow \mathbb{R}$	$h(x) \in \mathbb{R}^n$	$h \leftarrow x \overset{n}{\leftarrow}$	(Element-wise function)
$u : \mathbb{R}^n \rightarrow \mathbb{R}^m$ $v \in \mathbb{R}^m$	$v^T u(x) \in \mathbb{R}$	$u \overset{m}{\leftarrow} v \leftarrow^n x$	(Vector times vector function)
$A : \mathbb{R}^n \rightarrow \mathbb{R}^{m \times n}$, $v \in \mathbb{R}^n$	$A(x)v \in \mathbb{R}^m$	$\overset{m}{A} \overset{n}{\leftarrow} v \leftarrow^n x$	(Vector times matrix function)
$f : \mathbb{R}^d \rightarrow \mathbb{R}$, $X \in \mathbb{R}^{b \times d}$	$f(X) \in \mathbb{R}^b$	$f \overset{d}{\leftarrow} X \overset{b}{\leftarrow}$	(Vector function, batched)
$A : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}$, $X \in \mathbb{R}^{n \times m}$	$A(X) \in \mathbb{R}$	$A \overset{n}{\leftarrow} \overset{m}{\leftarrow} X$	(Matrix input function)
$f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^d$, $u \in \mathbb{R}^{b \times n}, v \in \mathbb{R}^{b \times m}$	$f(u, v) \in \mathbb{R}^{b \times d}$	$\overset{d}{f} \overset{n}{\leftarrow} \overset{m}{\leftarrow} u \overset{b}{\leftarrow} v$	(Two inputs, batched)

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

$$\text{pow}_n \overbrace{\leftarrow^n}^m u \overbrace{\leftarrow^n}^m v \multimap = \text{pow}_{n+m} \overbrace{\leftarrow^{n+m}}^m u \multimap$$

For a more advanced example, consider the softmax function: $\frac{d}{dx} \text{softmax} \overbrace{\leftarrow^n}^d x$. We can write this using elementary functions as:

$$\text{softmax}(x) = \frac{\exp(x)}{\text{sum}(\exp(x))} = \frac{\exp \overbrace{\leftarrow^n}^d x \multimap}{\text{pow}_{-1} \overbrace{\leftarrow^n}^d (\exp \overbrace{\leftarrow^n}^d x \multimap)}$$

It looks a bit complicated at first, but let us break it down step by step: $(\exp \overbrace{\leftarrow^n}^d x \multimap) \in \mathbb{R}^n$ is the element-wise exponential function. If we contract it with \multimap , 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 if 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 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and a matrix $x \in \mathbb{R}^{m \times n}$:

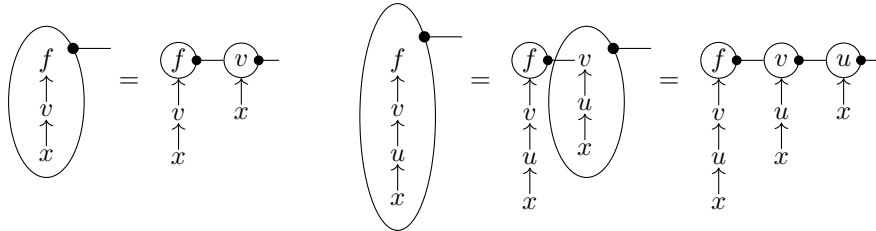
$$f \overbrace{\leftarrow^n}^m x = \overbrace{f(x)}^m = \text{Tr}(f(x)).$$

4.1 The Chain Rule

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

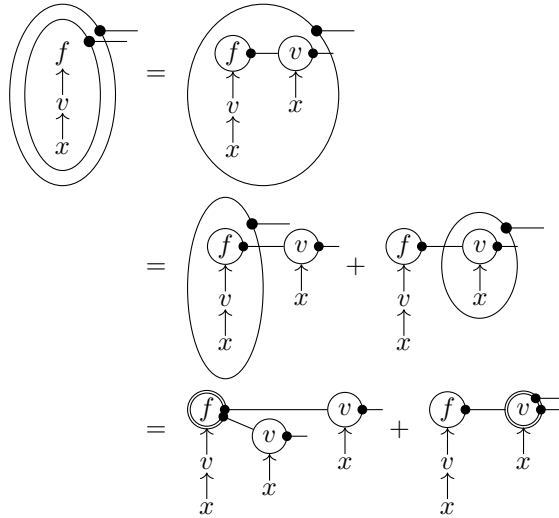
We could further simplify \textcircled{f} to f' —, where f' is the derivative of f . For example, if f was \cos —, then f' would be \sin —. But we can also just keep the circled f as an easy to recognize symbol for the derivative.

4.1.1 The Hessian Chain Rule

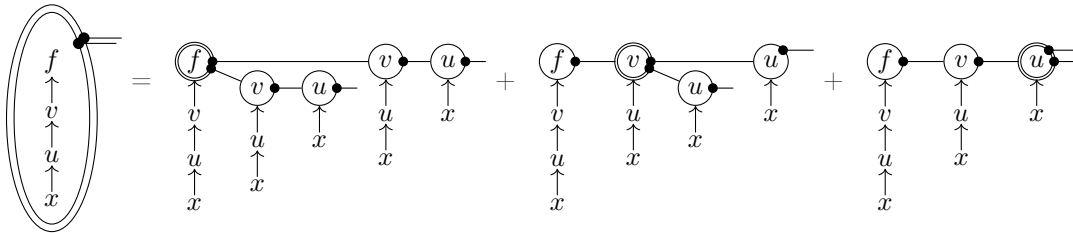
Things get even more interesting when we consider the second derivative. Using standard notation, the Hessian of $f \circ v$ is

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

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



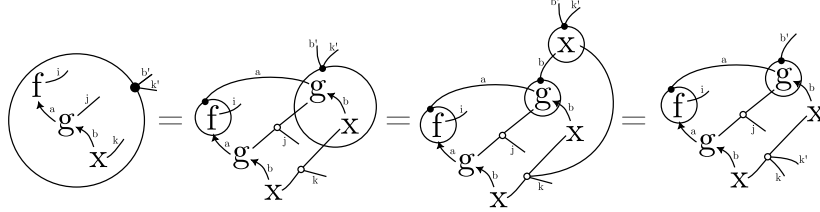
If we continue to expand this way, we see there is a simple pattern in the chain rule for more functions:



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.

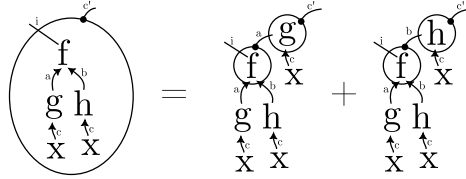
4.1.2 Chain Rule with Broadcasting

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

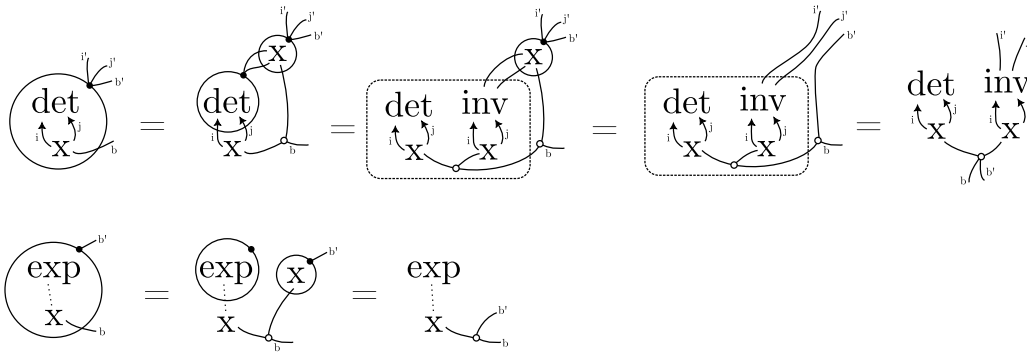
If $f : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}$ is a function with two inputs, the derivative of $f(x, y)$ is simply $\frac{\partial f}{\partial x} \partial x + \frac{\partial f}{\partial y} \partial y$, or in tensor diagram notation:



4.2 Examples

4.2.1 Known derivatives

Two examples where we can expand $(f) \bullet$ with known derivatives:



Here $\text{inv}(X)$ is the matrix inverse function, taking a matrix X to its inverse X^{-1} . In the case of the determinant, one may continue taking derivatives to find the simple pattern

$$\frac{\partial^k \det(X)}{\partial X^k} = \det(X)(X^{-1})^{\otimes k}.$$

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:

$$\begin{array}{c} \text{---} A \text{---} x \\ \uparrow \\ x \end{array} = \begin{array}{c} \text{---} A \text{---} x \\ \uparrow \\ x \end{array} + \begin{array}{c} \text{---} A \text{---} x \\ \uparrow \\ x \end{array} = \begin{array}{c} \text{---} A \text{---} x \\ \uparrow \\ x \end{array} + \begin{array}{c} \text{---} A \text{---} \\ \uparrow \\ x \end{array}$$

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

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

Which finally implies:

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

4.2.3 Trace identity

The Matrix Cookbook has the formula:

$$\frac{\partial \text{Tr}(\sin(x))}{\partial x} = \cos(x)^T \quad (128)$$

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

$$\begin{array}{c} \text{sin} \\ \vdots \\ x \end{array} = \begin{array}{c} \text{sin} \\ \vdots \\ x \end{array} + \begin{array}{c} \text{sin} \\ \vdots \\ x \end{array} = \begin{array}{c} \text{cos} \\ \vdots \\ x \end{array} + \begin{array}{c} \text{cos} \\ \vdots \\ x \end{array}$$

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 $\text{Tr}(F(X))$ 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 an n -times differentiable function $v : \mathbb{R}^d \rightarrow \mathbb{R}^d$ we can write the Taylor expansion:

$$\begin{array}{c} v \\ \uparrow \\ (x + \varepsilon) \end{array} \approx \begin{array}{c} v \\ \uparrow \\ x \end{array} + \begin{array}{c} \varepsilon \\ \bullet \\ \uparrow \\ x \end{array} + \frac{1}{2} \begin{array}{c} \varepsilon \\ \bullet \\ \varepsilon \\ \bullet \\ \uparrow \\ x \end{array} + \frac{1}{6} \begin{array}{c} \varepsilon \\ \bullet \\ \varepsilon \\ \bullet \\ \varepsilon \\ \bullet \\ \uparrow \\ x \end{array} + \dots$$

Writing this using classical notation is quite messy:

$$\begin{aligned} v(x + \varepsilon) &\approx v(x) + \left[\frac{\partial}{\partial x} v(x) \right] \varepsilon + \frac{1}{2} \left[\frac{\partial}{\partial x} \left[\frac{\partial}{\partial x} v(x) \right] \varepsilon \right] \varepsilon + \frac{1}{6} \left[\frac{\partial}{\partial x} \left[\frac{\partial}{\partial x} \left[\frac{\partial}{\partial x} v(x) \right] \varepsilon \right] \varepsilon \right] \varepsilon + \dots \\ &= v(x) + \left[\frac{\partial}{\partial x} v(x) \right] \varepsilon + \frac{1}{2} (I \otimes \varepsilon) \left[\frac{\partial \text{vec}}{\partial x} \left[\frac{\partial v(x)}{\partial x} \right] \right] \varepsilon + \frac{1}{6} (I \otimes \varepsilon \otimes \varepsilon) \left[\frac{\partial \text{vec}}{\partial x} \left[\frac{\partial \text{vec}}{\partial x} \left[\frac{\partial v(x)}{\partial x} \right] \right] \right] \varepsilon + \dots \end{aligned}$$

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 20. Draw the tensor diagram for a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that applies an element-wise 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 21. Represent the composition of two functions

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

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

$$KL(\mu, \sigma) = -\frac{1}{2} \left(1 + \log \sigma^2 - \mu^2 - \sigma^2 \right),$$

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 24. Let the softmax function $s : \mathbb{R}^n \rightarrow \mathbb{R}^n$ 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

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

where δ_{ij} is the Kronecker delta.

Exercise 25. 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 26. Consider a Gaussian process with covariance matrix $K(\theta)$ and the log-marginal likelihood defined as

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

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

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

Exercise 27. 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

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

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 28. 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 29. 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 30. 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 31. 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 32. 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 33. 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 34. 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

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

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

Exercise 35. (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 36. 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

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

where $\text{diag}(s)$ 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 37. Let $A \in \mathbb{R}^{p \times m}$, $X \in \mathbb{R}^{m \times n}$, and $B \in \mathbb{R}^{n \times q}$. Prove the identity

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

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

Exercise 38. 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 39. Consider the Frobenius norm $\|X\|_F = \sqrt{\text{Tr}(X^T X)}$, for $X \in \mathbb{R}^{m \times n}$.

Show that

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

for $X \neq 0$.

Exercise 40. 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 $k > 1$, show that the derivative of $\phi(Y)$ is not unique. In particular, demonstrate that any matrix of the form

$$G = \sum_{i=1}^k u_i v_i^T,$$

where $\{u_i\}_{i=1}^k$ is any orthonormal basis for the eigenspace corresponding to the largest eigenvalue, is a valid subgradient of $\phi(Y)$. Explain the challenges that arise in defining a unique gradient in this setting.

Chapter 5

Statistics and Probability

5.1 Definition of Moments

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

In tensor diagrams, we will use square brackets:

$$-m = [-x] \quad \text{and} \quad -M- = [-(x \ominus m) \quad (x \ominus m)-]$$

We will use the circled minus, \ominus , to distinguish the operation from contraction edges.

We can also define the third and fourth centralized moment tensors

$$M_3 \begin{array}{c} \diagup \\ \diagdown \end{array} = \begin{bmatrix} (x \ominus m)- \\ (x \ominus m)- \\ (x \ominus m)- \end{bmatrix} \quad \text{and} \quad M_4 \begin{array}{c} \diagup \\ \diagdown \end{array} = \begin{bmatrix} (x \ominus m)- \\ (x \ominus m)- \\ (x \ominus m)- \\ (x \ominus m)- \end{bmatrix}.$$

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 .

$$\left[\begin{array}{ccccc} A & & X & - & B & & C \\ | & \diagup & | & & & \diagup & | \\ X & - & D & - & X & - & E \end{array} \right] = \begin{array}{ccccc} A & & X & - & B & & C \\ | & \diagup & | & & & \diagup & | \\ \boxed{X} & - & \boxed{D} & - & \boxed{X} & - & \boxed{E} \end{array} = \begin{array}{c} A \quad \bigcap \\ \diagdown \quad M_3 \quad \diagup \\ \bigcap \quad D \quad \bigcap \end{array} \begin{array}{cc} B & C \\ | & | \\ E & \end{array}$$

where M_3 is the expectation¹ $\left[\begin{array}{c} | \quad | \quad | \\ \diagdown \quad \diagup \quad \diagdown \end{array} \right]$, which is an order-9 tensor with no dependence on the constants A, B, C and D . In practice you would want to name the edges to keep track of what gets multiplied with what.

¹FIXME: This is different from the notation just introduced above.

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:

$$\begin{aligned} E(e^{\langle x \ominus m, t \rangle}) &= \sum_{k=0}^{\infty} \frac{1}{k!} E[\langle x \ominus m, t \rangle^k] = E\left[\sum_{k=0}^{\infty} \frac{1}{k!} \langle (x \ominus m)^{\otimes k}, t^{\otimes k} \rangle\right] = \sum_{k=0}^{\infty} \frac{1}{k!} \langle M_k, t^{\otimes k} \rangle \\ &= 1 + \frac{t}{m} + \frac{1}{2} \overset{t}{\underset{t}{\parallel}} M + \frac{1}{6} \overset{t}{\underset{t}{\parallel}} M_3 + \frac{1}{24} \overset{t}{\underset{t}{\parallel}} M_4 + \dots \end{aligned}$$

5.1.2 Linear Forms

The Matrix Cookbook gives the following simple expectation:

$$E[AXB + C] = AE[X]B + C \quad \left[\begin{array}{c} -A-X-B- \\ + -C- \end{array} \right] = \begin{array}{c} -A-[X]-B- \\ + -C- \end{array} \quad (312)$$

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:

$$\left[\begin{array}{c} x- \\ x- \end{array} \right] = \left[\begin{array}{c} (x \ominus m)- \\ (x \ominus m)- \end{array} \right] + \frac{m-}{m-} = M \prec + \frac{m-}{m-}$$

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

$$\begin{aligned} \text{Var}[Ax] &= A \text{Var}[x] A^T \quad \left[\begin{array}{c} A-x \ominus [A-x] \\ A-x \ominus [A-x] \end{array} \right] = \left[\begin{array}{c} A-(x \ominus m) \\ A-(x \ominus m) \end{array} \right] \\ &= A \left[\begin{array}{c} (x \ominus m) \\ (x \ominus m) \end{array} \right] \\ &= -A-M_2-A- \end{aligned} \quad (313)$$

$$\begin{aligned} E[x^T Ax] &= \text{Tr}(AM) + m^T A m \quad [x-A-x] = \left(\left[\begin{array}{c} (x \ominus m)- \\ (x \ominus m)- \end{array} \right] + \frac{m-}{m-} \right) \succ A \\ &= \left[\begin{array}{c} (x \ominus m) \\ (x \ominus m) \end{array} \right] \succ A + \frac{m \succ A}{m \succ A} \\ &= \overbrace{M-A} + m-A-m \end{aligned} \quad (318)$$

$$E[(A\mathbf{x} + a)(B\mathbf{x} + b)^T] = AMB^T + (Am + a)(Bm + b)^T \quad (320)$$

$$E[\mathbf{x}\mathbf{x}^T] = M + mm^T \quad (321)$$

$$E[\mathbf{x}a^T\mathbf{x}] = (M + mm^T)a \quad (322)$$

$$E[\mathbf{x}^T a\mathbf{x}^T] = a^T(M + mm^T) \quad (323)$$

$$E[(A\mathbf{x})(A\mathbf{x})^T] = A(M + mm^T)A^T \quad (324)$$

$$E[(\mathbf{x} + a)(\mathbf{x} + a)^T] = M + (m + a)(m + a)^T \quad (325)$$

$$E[(A\mathbf{x} + a)^T(B\mathbf{x} + b)] = \text{Tr}(AMB^T) + (Am + a)^T(Bm + b) \quad (326)$$

$$E[\mathbf{x}^T\mathbf{x}] = \text{Tr}(M) + m^T m \quad (327)$$

$$E[\mathbf{x}^T A\mathbf{x}] = \text{Tr}(AM) + m^T Am \quad (328)$$

$$E[(A\mathbf{x})^T(A\mathbf{x})] = \text{Tr}(AMA^T) + (Am)^T(Am) \quad (329)$$

$$E[(\mathbf{x} + a)^T(\mathbf{x} + a)] = \text{Tr}(M) + (m + a)^T(m + a) \quad (330)$$

5.1.4 Cubic Forms



When x is a stochastic vector with mean vector m , it can be convenient to expand the raw third moment in terms of the central moments:

$$\begin{aligned} \begin{bmatrix} x- \\ x- \\ x- \end{bmatrix} &= \begin{bmatrix} (x \ominus m)- \\ (x \ominus m)- \\ (x \ominus m)- \end{bmatrix} + 3 \begin{bmatrix} (x \ominus m)- \\ m- \\ m- \end{bmatrix} + 3 \begin{bmatrix} m- \\ (x \ominus m)- \\ (x \ominus m)- \end{bmatrix} + \begin{bmatrix} m- \\ m- \\ m- \end{bmatrix} \\ &= \overset{|}{\underset{/\backslash}{M_3}} + 3 \begin{matrix} m- \\ -M- \end{matrix} + \begin{matrix} m- \\ m- \\ m- \end{matrix} \end{aligned}$$

TODO: The edges from the m, M term needs to be symmetrized.

But this is still a bit of a mess. See also below on Cumulants.

Assume \mathbf{x} to be a stochastic vector with independent coordinates, mean m , covariance M and central moments $v_3 = \mathbb{E}[(\mathbf{x} - m)^3]$. Then (see [7])

$$\begin{aligned} &\mathbb{E}[(A\mathbf{x} + a)(B\mathbf{x} + b)^T(C\mathbf{x} + c)] \\ &= A \text{diag}(B^T C) v_3 \\ &+ (Am + a) \text{Tr}(BMC^T) \\ &+ AMC^T(Bm + b) \\ &+ AMB^T(Cm + c) \\ &+ (Am + a)(Bm + b)^T(Cm + c) \\ &\mathbb{E}[\mathbf{x}\mathbf{x}^T\mathbf{x}] \\ &= v_3 \\ &+ 2Mm \end{aligned}$$

$$+ (\text{Tr}(M) + m^T m)m$$

5.2 Cumulants

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

$$\log E(e^{\langle t, x \rangle}) = \sum_{n=1}^{\infty} \frac{1}{n!} \langle K_n, t^{\otimes n} \rangle = \frac{t}{K_1} + \frac{1}{2} \frac{t}{K_2} + \frac{1}{6} \frac{t}{K_3} + \dots$$

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

$$K_1 = m \quad \text{and} \quad K_2 = M \quad \text{and} \quad K_3 = M_3 \quad \text{and} \quad K_4 = M_4 - \frac{-M_2^-}{-M_2^-} - \frac{M_2^-}{M_2^-} - \frac{M_2^-}{M_2^-}.$$

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:

$$\left[\begin{array}{c} x \quad x \\ x \\ | \end{array} \right] = \frac{K_3}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_1^-}{K_2} + \frac{K_1^-}{K_1}.$$

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

If the entries of x are *independent*, the off-diagonals of the cumulant tensors K_1, K_2, \dots are zero. This means Assume each x_i has cumulants $\kappa_1, \kappa_2, \kappa_3, \kappa_4 \in \mathbb{R}$, then

$$\left[\begin{array}{c} x \quad x \\ x \\ | \end{array} \right] = \begin{array}{l} \kappa_4 \frac{K_4}{K_1} \\ + \kappa_3 \kappa_1 \left(\frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} \right) \\ + \kappa_2^2 \left(\frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} \right) \\ + \kappa_2 \kappa_1^2 \left(\frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} \right) \\ + \kappa_1^4 \frac{K_1^-}{K_1} \end{array}$$

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 $\kappa_1 = 0$:

$$\kappa_5 \frac{K_5}{K_1} + \kappa_3 \kappa_2 \left(\frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} + \frac{K_3^-}{K_1} \right)$$

If x is Gaussian, all cumulants of order 3 and higher are zero. For $n = 6$ there are 203 partitions in total, but only 15 terms² for $E[x^{\otimes 6}]$:

$$\begin{array}{l} \kappa_2^3 \left(\frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} \right) \\ + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} + \frac{K_2^-}{K_1} \end{array}.$$

²This is why $E[g^6] = 15$ for $g \sim N(0, 1)$.

5.2.1 Quartic Forms

We can use this to compute $\text{Var}[x^T A x]$. We assume A is symmetric:

$$\begin{aligned}
 \text{Var}[x^T A x] &= \boxed{\begin{array}{c} x \quad A \quad x \\ x \quad A \quad x \end{array}} - \frac{\boxed{\begin{array}{c} x \quad A \quad x \\ x \quad A \quad x \end{array}}}{\boxed{\begin{array}{c} x \quad A \quad x \\ x \quad A \quad x \end{array}}} \\
 &= \kappa_4 A \frown A + 4\kappa_3 \kappa_1 A \frown A - \circ \\
 &\quad + 2\kappa_2^2 \overbrace{A-A} + \kappa_2^2 \overbrace{A} \overbrace{A} \\
 &\quad + 4\kappa_2 \kappa_1^2 \circ - A - A - \circ + 2\kappa_2 \kappa_1^2 \overbrace{A} \circ - A - \circ \\
 &\quad + \kappa_1^4 \circ - A - \circ - A - \circ \\
 &\quad - (\kappa_2 \overbrace{A} + \kappa_1^2 \circ - A - \circ)^2 \\
 &= 2\kappa_2^2 \overbrace{A-A} + 4\kappa_2 \kappa_1^2 \circ - A - A - \circ \\
 &\quad + 4\kappa_3 \kappa_1 A \frown A - \circ + \kappa_4 A \frown A
 \end{aligned}$$

The Matrix Cookbook lists this as

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

where $c = \mu_1$ is the mean of x , and $a = \text{diag}(A)$ is the diagonal of A , and $\mu_4 = \kappa_4 + 3\kappa_2^2$ is the fourth central moment.

5.3 Weighted Scalar Variable

Let $y = w^T x$, and let $m = E[y]$, then

$$E[y] = m = w^T \mu \quad (321)$$

$$E[(y - m)^2] = w^T M_2 w - m^2 \quad (322)$$

$$E[(y - m)^3] = w^T M_3 w - 3m w^T M_2 w + 3m^2 w^T \mu - m^3 \quad (323)$$

$$E[(y - m)^4] = w^T M_4 w - 4m w^T M_3 w + 6m^2 w^T M_2 w - 3m^4 \quad (324)$$

For $x \sim N(0, 1)$, we have the inequality:

$$E[y^n]^{1/n} \leq \sqrt{2/\pi} \|w\|_2.$$

5.4 Gaussian Moments

For a Gaussian vector $x \sim \mathcal{N}(m, M)$: - All odd centered moments vanish: $M_3 = 0$, etc. - Even moments can be computed via Isserlis' theorem.

For instance:

$$\mathbb{E}[(x \ominus m)^{\otimes 4}] = M \otimes M + M \otimes M + \dots$$

(summing over the different pairings of indices).

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.

$$E(xx^T) = \Sigma + mm^T \quad (377)$$

$$E[x^T Ax] = \text{Tr}(A\Sigma) + m^T Am \quad (378)$$

$$\begin{aligned} \text{Var}(x^T Ax) &= \text{Tr}[A\Sigma(A + A^T)\Sigma] + \dots \\ &\quad + m^T(A + A^T)\Sigma(A + A^T)m \end{aligned} \quad (379)$$

$$E[(x - m')^T A(x - m')] = (m - m')^T A(m - m') + \text{Tr}(A\Sigma) \quad (380)$$

If $\Sigma = \sigma^2 I$ and A is symmetric, then

$$\text{Var}(x^T Ax) = 2\sigma^4 \text{Tr}(A^2) + 4\sigma^2 m^T A^2 m \quad (381)$$

Assume $x \sim \mathcal{N}(0, \sigma^2 I)$ and A and B to be symmetric, then

$$\text{Cov}(x^T Ax, x^T Bx) = 2\sigma^4 \text{Tr}(AB) \quad (382)$$

5.4.2 Cubic forms

Assume x to be a stochastic vector with independent coordinates, mean m and covariance M

$$\begin{aligned} E[xb^T xx^T] &= mb^T(M + mm^T) + (M + mm^T)bm^T \\ &\quad + b^T m(M - mm^T) \end{aligned} \quad (383)$$

5.4.3 Mean of Quartic Forms

$$E[xx^T xx^T] = 2(\Sigma + mm^T)^2 + m^T m(\Sigma - mm^T)$$

$$+ \text{Tr}(\Sigma)(\Sigma + mm^T)$$

$$\begin{aligned} E[xx^T Axx^T] &= (\Sigma + mm^T)(A + A^T)(\Sigma + mm^T) \\ &\quad + m^T Am(\Sigma - mm^T) + \text{Tr}[A\Sigma](\Sigma + mm^T) \end{aligned}$$

$$E[x^T xx^T x] = 2\text{Tr}(\Sigma^2) + 4m^T \Sigma m + (\text{Tr}(\Sigma) + m^T m)^2$$

$$\begin{aligned} E[x^T Axx^T Bx] &= \text{Tr}[A\Sigma(B + B^T)\Sigma] + m^T(A + A^T)\Sigma(B + B^T)m \\ &\quad + (\text{Tr}(A\Sigma) + m^T Am)(\text{Tr}(B\Sigma) + m^T Bm) \end{aligned}$$

$$\begin{aligned} E[a^T xb^T xc^T xd^T x] &= (a^T(\Sigma + mm^T)b)(c^T(\Sigma + mm^T)d) \\ &\quad + (a^T(\Sigma + mm^T)c)(b^T(\Sigma + mm^T)d) \\ &\quad + (a^T(\Sigma + mm^T)d)(b^T(\Sigma + mm^T)c) - 2a^T mb^T mc^T md^T m \end{aligned}$$

$$\begin{aligned} &E[(Ax + a)(Bx + b)^T(Cx + c)(Dx + d)^T] \\ &= [A\Sigma B^T + (Am + a)(Bm + b)^T][C\Sigma D^T + (Cm + c)(Dm + d)^T] \\ &\quad + [A\Sigma C^T + (Am + a)(Cm + c)^T][B\Sigma D^T + (Bm + b)(Dm + d)^T] \\ &\quad + (Bm + b)^T(Cm + c)[A\Sigma D^T - (Am + a)(Dm + d)^T] \\ &\quad + \text{Tr}(B\Sigma C^T)[A\Sigma D^T + (Am + a)(Dm + d)^T] \end{aligned}$$

$$\begin{aligned} &E[(Ax + a)^T(Bx + b)(Cx + c)^T(Dx + d)] \\ &= \text{Tr}[A\Sigma(C^T D + D^T C)\Sigma B^T] \\ &\quad + [(Am + a)^T B + (Bm + b)^T A]\Sigma[C^T(Dm + d) + D^T(Cm + c)] \\ &\quad + [\text{Tr}(A\Sigma B^T) + (Am + a)^T(Bm + b)][\text{Tr}(C\Sigma D^T) + (Cm + c)^T(Dm + d)] \end{aligned}$$

See [7].

5.4.4 Mixture of Gaussians

For a mixture of Gaussians:

$$x \sim \sum_k \pi_k \mathcal{N}(m_k, M_k),$$

the moments are weighted sums:

$$E[x] = \sum_k \pi_k m_k, \quad \text{Var}[x] = \sum_k \pi_k (M_k + m_k m_k^T) - \left(\sum_k \pi_k m_k \right) \left(\sum_k \pi_k m_k \right)^T.$$

Higher moments similarly combine via linearity.

$$E[x] = \sum_k \rho_k m_k \quad (384)$$

$$\text{Cov}(x) = \sum_k \sum_{k'} \rho_k \rho_{k'} (\Sigma_k + m_k m_k^T - m_k m_{k'}^T) \quad (385)$$

5.4.5 Derivatives

Derivatives of moments with respect to m or M can be found by differentiating under the integral sign, and using Stein's lemma for Gaussian cases.

5.5 Exercises

Exercise 41.

$$\begin{aligned} \mathbb{E}[(A\mathbf{x} + a)(A\mathbf{x} + a)^T(A\mathbf{x} + a)] &= \text{Adiag}(A^T A)v_3 \\ &+ [2AMA^T + (A\mathbf{x} + a)(A\mathbf{x} + a)^T](Am + a) \\ &+ \text{Tr}(AMA^T)(Am + a) \end{aligned}$$

$$\begin{aligned} \mathbb{E}[(A\mathbf{x} + a)b^T(C\mathbf{x} + c)(D\mathbf{x} + d)^T] &= (A\mathbf{x} + a)b^T(CMD^T + (Cm + c)(Dm + d)^T) \\ &+ (AMC^T + (Am + a)(Cm + c)^T)b(Dm + d)^T \\ &+ b^T(Cm + c)(AMD^T - (Am + a)(Dm + d)^T) \end{aligned}$$

Exercise 42. Find more identities in the Matrix Reference Manual and try to prove them. Also try to verify your derivations using tensorgrad.

Chapter 6

Determinant and Inverses

6.1 Determinant

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

$$\det(A) = \frac{1}{n!} \overline{\overline{A \cdots A}}$$

where $\overline{\overline{i_1 \ i_2 \ \cdots \ i_n}} = \varepsilon_{i_1, \dots, i_n}$ is the rank- n Levi-Civita tensor defined by

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

To see that the definition makes sense, let's first consider

$$\det(I) = \frac{1}{n!} \overline{\overline{1 \cdots 1}} = \frac{1}{n!} \sum_{i_1, \dots, i_n, j_1, \dots, j_n} \varepsilon_{i_1, \dots, i_n} \varepsilon_{j_1, \dots, j_n} [i = j] = \frac{1}{n!} \sum_{i_1, \dots, i_n} \varepsilon_{i_1, \dots, i_n}^2 = 1.$$

In general we get from the permutation definition of the determinant:

$$\begin{aligned} \overline{\overline{A \cdots A}} &= \sum_{i_1, \dots, i_n, j_1, \dots, j_n} \varepsilon_{i_1, \dots, i_n} \varepsilon_{j_1, \dots, j_n} A_{i_1, j_1} \cdots A_{i_n, j_n} \\ &= \sum_{\sigma, \tau} \text{sign}(\sigma) \text{sign}(\tau) A_{\sigma_1, \tau_1} \cdots A_{\sigma_n, \tau_n} \\ &= \sum_{\sigma} \text{sign}(\sigma) \sum_{\tau} \text{sign}(\tau) A_{\sigma_1, \tau_1} \cdots A_{\sigma_n, \tau_n} \\ &= \sum_{\sigma} \text{sign}(\sigma)^2 \det(A) \\ &= n! \det(A). \end{aligned}$$

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

A curious property is that

$$\begin{array}{|c|} \hline A \\ \hline \end{array} \cdots \begin{array}{|c|} \hline A \\ \hline \end{array} = \begin{array}{|c|} \hline \cdots \\ \hline \end{array} \begin{array}{|c|} \hline A \\ \hline \end{array} \cdots \begin{array}{|c|} \hline A \\ \hline \end{array}$$

$$(18) \quad \det(A) = \prod_i \lambda_i \quad \dots$$

$$(19) \quad \det(cA) = c^n \det(A) \quad \begin{array}{|c|} \hline cA \\ \hline \end{array} \cdots \begin{array}{|c|} \hline cA \\ \hline \end{array} = c^n \begin{array}{|c|} \hline A \\ \hline \end{array} \cdots \begin{array}{|c|} \hline A \\ \hline \end{array}$$

$$(20) \quad \det(A) = \det(A^T) \quad \dots$$

$$(21) \quad \det(AB) = \det(A)\det(B) \quad \begin{array}{|c|} \hline A \\ \hline \end{array} \cdots \begin{array}{|c|} \hline A \\ \hline \end{array} \begin{array}{|c|} \hline B \\ \hline \end{array} \cdots \begin{array}{|c|} \hline B \\ \hline \end{array} = \begin{array}{|c|} \hline A \\ \hline \end{array} \cdots \begin{array}{|c|} \hline A \\ \hline \end{array} \begin{array}{|c|} \hline B \\ \hline \end{array} \cdots \begin{array}{|c|} \hline B \\ \hline \end{array}$$

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

$$(23) \quad \det(A^n) = \det(A)^n \quad \dots$$

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

6.2 Inverses

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

Chapter 7

Advanced Derivatives

7.1 Derivatives of vector norms

7.1.1 Two-norm

$$\frac{d}{dx} \|x - a\|_2 = \frac{x - a}{\|x - a\|_2} \quad (7.1)$$

$$\frac{d}{dx} \frac{x - a}{\|x - a\|_2} = \frac{I}{\|x - a\|_2} - \frac{(x - a)(x - a)^T}{\|x - a\|_2^3} \quad (7.2)$$

$$\frac{d}{dx} \|x\|_2^2 = \frac{d}{dx} \|x^T x\|_2 = 2x \quad (7.3)$$

7.2 Derivatives of matrix norms

7.3 Derivatives of Structured Matrices

7.3.1 Symmetric

7.3.2 Diagonal

7.3.3 Toeplitz

7.4 Derivatives of a Determinant

7.5 General forms

7.6 Linear forms

7.7 Square forms

7.8 From Stack Exchange

Let $F(t) = |1 + tA|$ then $F'(t) = \text{Tr}(A)$ and $F''(t) = \text{Tr}(A)^2 - \text{Tr}(A^2)$.

7.9 Derivatives of an Inverse

7.9.1 Trace Identities

$$\frac{\partial}{\partial \mathbf{X}} \text{Tr}(\mathbf{A}\mathbf{X}^{-1}\mathbf{B}) = -(\mathbf{X}^{-1}\mathbf{B}\mathbf{A}\mathbf{X}^{-1})^T = -\mathbf{X}^{-T}\mathbf{A}^T\mathbf{B}^T\mathbf{X}^{-T}$$

Assume \mathbf{B} and \mathbf{C} to be symmetric, then

$$\begin{aligned} \frac{\partial}{\partial \mathbf{X}} \text{Tr}[(\mathbf{X}^T\mathbf{C}\mathbf{X})^{-1}\mathbf{A}] &= -(\mathbf{C}\mathbf{X}(\mathbf{X}^T\mathbf{C}\mathbf{X})^{-1})(\mathbf{A} + \mathbf{A}^T)(\mathbf{X}^T\mathbf{C}\mathbf{X})^{-1} \\ \frac{\partial}{\partial \mathbf{X}} \text{Tr}[(\mathbf{X}^T\mathbf{C}\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{B}\mathbf{X})] &= -2\mathbf{C}\mathbf{X}(\mathbf{X}^T\mathbf{C}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{B}\mathbf{X}(\mathbf{X}^T\mathbf{C}\mathbf{X})^{-1} \\ \frac{\partial}{\partial \mathbf{X}} \text{Tr}[(\mathbf{A} + \mathbf{X}^T\mathbf{C}\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{B}\mathbf{X})] &= -2\mathbf{B}\mathbf{X}(\mathbf{X}^T\mathbf{C}\mathbf{X})^{-1} \\ &\quad - 2\mathbf{C}\mathbf{X}(\mathbf{A} + \mathbf{X}^T\mathbf{C}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{B}\mathbf{X}(\mathbf{A} + \mathbf{X}^T\mathbf{C}\mathbf{X})^{-1} \\ &\quad + 2\mathbf{B}\mathbf{X}(\mathbf{A} + \mathbf{X}^T\mathbf{C}\mathbf{X})^{-1} \end{aligned}$$

7.10 Derivatives of Eigenvalues

7.11 Exercises

Exercise 43. Find the derivative of

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

with respect to Σ .

Chapter 8

Special Matrices

8.0.1 Block matrices

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

8.0.2 The Discrete Fourier Transform Matrix

I think FFT can be nicely described with diagrams

Let's start with the Hadamard matrix: $H_n = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^{\otimes n}$. Hm. It's just a bunch of matrices below each other, kinda boring.

What about the FFT? Does that require a bit more?

8.0.3 Fast Kronecker Multiplication

Say we want to compute $(A_1 \otimes A_2 \cdots A_n)x$, where A_i is a $a_i \times a_i$ 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 a $a_1 \times \cdots a_n$ tensor and perform the multiplication

$$\begin{array}{c} \xrightarrow{a_1} A_1 \\ \xrightarrow{a_2} A_2 \\ \vdots \\ \xrightarrow{a_n} A_n \end{array} \begin{array}{c} \searrow^{a_1} \\ \xrightarrow{\quad} \\ \nearrow_{a_n} \end{array} X$$

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 algorithm as we will see.

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

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

$$H_{2^n} = \begin{array}{c} \text{---} \triangleleft \begin{array}{c} H_2 \\ \vdots \\ H_7 \end{array} \triangleright \text{---} \end{array}.$$
$$H_{2^n}x = \begin{bmatrix} H_{2^{n-1}} & H_{2^{n-1}} \\ H_{2^{n-1}} & -H_{2^{n-1}} \end{bmatrix} \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix},$$

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 $(a_1 a_2 \cdots a_n)(a_1 + a_2 + \cdots a_n) = 2n \log_2 n$.

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

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

$$= \left\{ \exp \frac{2\pi i}{N} \begin{matrix} \text{arange}(N) \\ \text{arange}(N) \end{matrix} \right\}$$

TODO: Show how the matrix can be written with the function notation.

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

$$F_N = \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{P_1} \\ \vdots \\ \boxed{P_1} \end{array} \begin{array}{c} F_{p_1^{i_1}} \\ F_{p_2^{i_2}} \\ \vdots \\ F_{p_n^{i_n}} \end{array} \begin{array}{c} \boxed{P_2} \\ \vdots \\ \boxed{P_2} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array},$$

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, the algorithm 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 $\sim n^2/\log n = O(\log(N)^2)$. 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 $D_N = [1, w^N, w^{2N}, \dots]$. The even-odd permutation moves all the even values to the start. If we reshape I_{2^n} as $I_2 \otimes \cdots \otimes I_2$, this permutation is just $P_N = \text{---}$, or in pytorch: `x.permute([3,0,1,2])`. Also note that $\begin{bmatrix} I & I \\ I & -I \end{bmatrix} = H_2 \otimes I$ and $\begin{bmatrix} F_{N/2} & 0 \\ 0 & F_{N/2} \end{bmatrix} = I_2 \otimes F_{N/2}$. So we can write in tensor diagram notation:

$$F_N = \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{H_2} \\ \vdots \\ \boxed{H_2} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{F_{N/2}} \\ \vdots \\ \boxed{F_{N/2}} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} = \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{H_2} \\ \vdots \\ \boxed{H_2} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{H_2} \\ \vdots \\ \boxed{H_2} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{H_2} \\ \vdots \\ \boxed{H_2} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array}$$

$D_N \qquad D_{N/2^0} \quad D_{N/2^1} \quad D_{N/2^2}$

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.

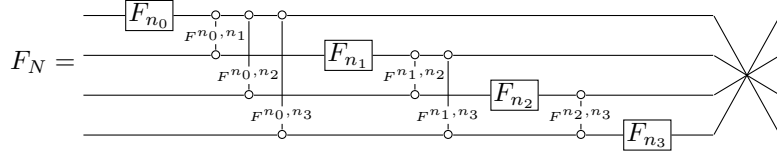
We don't have to split the matrix in half, we can also split it in thirds, fourths, etc. With this generalized Cooley-Tukey algorithm, we get the following diagram:

$$F_N = \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{F_{n_0}} \\ \vdots \\ \boxed{F_{n_0}} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{F_{n_1}} \\ \vdots \\ \boxed{F_{n_1}} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{F_{n_2}} \\ \vdots \\ \boxed{F_{n_2}} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{F_{n_3}} \\ \vdots \\ \boxed{F_{n_3}} \end{array} \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array}$$

$F^{(n_0, n_1 n_2 n_3)} F^{(n_1, n_2 n_3)} F^{(n_2, n_3)}$

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 $F_{j,k}^{(a,b)} = e^{-2\pi i j k / (ab)}$, 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.

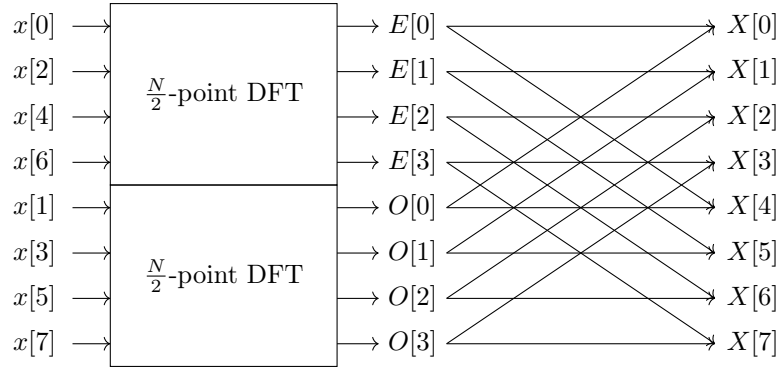
We can use the property $F_{a,bc} = F_{a,b} \bullet F_{a,c}^{1/b}$ to simplify the diagram further:



In the simple case where we split in 2 every time, this is also called the “Quantum FFT” algorithm.

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

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 $\log_2 N$).

Multi-dimensional Fourier Transform

This is just taking the Fourier transform along each axis.

8.0.4 Hermitian Matrices and skew-Hermitian

Complex. Skip

8.0.5 Idempotent Matrices

Skip

8.0.6 Orthogonal matrices

Skip

8.0.7 Positive Definite and Semi-definite Matrices

Skip

8.0.8 Singleentry Matrix, The

Describes the matrix J . All of this is trivial with diagrams.

8.0.9 Symmetric, Skew-symmetric/Antisymmetric

Could introduce Penrose's symmetric tensors here?

8.0.10 Toeplitz Matrices

Could talk about the convolution tensor here...

8.0.11 Units, Permutation and Shift

Not that interesting...

8.0.12 Vandermonde Matrices

Does this have a nice description? Not a lot of properties are given in the Cookbook.

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 . Here U is a square matrix, which we keep. We multiply the i th edge of A by U^T (which 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^T s onto A as we go along.

For order 3 tensors, this method is called a “Tucker decomposition”.

If the “core matrix” is diagonal, this 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.

I guess HOSVD gives a rank decomposition if we diagonalize the core tensor. It just won’t be an efficient one.

9.2 Rank Decomposition

9.2.1 Border Rank

The border rank of a tensor is the smallest rank of a tensor that is close to it. TODO: Example where the border rank is much smaller than the rank.

9.3 Fast Matrix Multiplication

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

$$S_A = \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \end{bmatrix}$$
$$S_B = \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, & \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, & \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \end{bmatrix}$$

$$W = \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, & \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, & \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, & \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, & \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \end{bmatrix}$$

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:

$$= A = B = \begin{array}{c} \text{---} W \text{---} \\ \text{---} S_A \text{---} S_B \text{---} \\ \text{---} A \text{---} B \text{---} \end{array}.$$

Contracting the edges in the right order, uses only $7/8n^3 + O(n^2)$ operations.

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

$$AB = \begin{array}{c} \text{---} A \text{---} B \text{---} \end{array},$$

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

Contracting the double edges, $S_A - A$ and $S_B - B$, is both $O(n^2)$ time.

It remains to verify that this is actually faster than the naive matrix multiplication: Contracting $S_A - A$ takes $7 \cdot 2^2(n/2)^2$ operations, and likewise $S_B - B$. Next we contract $S_A A - S_B B$ which takes $7(n/2)^3$ time. And finally we contract the edge with W which takes $2^2 \cdot 7(n/2)^2$. The important term is the cubic $7/8n^3$, which if instead done recursively, leads to the classical $O(n^{\log_2 7})$ algorithm.

FIXME: What “edge with W ”? I think we have to/want to contract the hyperedge with W immediately?

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, Deep Mind 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. This 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

Machine Learning Applications

10.1 Least Squares

10.2 Hessian of Cross Entropy Loss

10.3 Convolutional Neural Networks

10.4 Transformers / Attention

10.5 Tensor Sketch

10.6 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 11

Tensor Algorithms

11.1 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 :

$$\begin{array}{c} \text{---} \underset{A}{\overset{2}{\mid}} \text{---} \underset{4}{\mid} \underset{B}{\overset{2}{\mid}} \xrightarrow{\text{Contraction}} \text{---} \underset{A'}{\overset{2}{\mid}} \end{array}$$

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:

$$\begin{array}{c} \text{---} \underset{A}{\overset{2}{\mid}} \text{---} \underset{4}{\mid} \underset{B}{\overset{2}{\mid}} \text{---} \underset{1}{\mid} \underset{C}{\overset{2}{\mid}} \text{---} \underset{2}{\mid} \end{array}$$

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?*

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

11.1.1 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 [3]. There is a well-known exact algorithm running in $O(3^n)$ -time [17], 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 [22].

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 [4], 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 [26] (or even in $O(n \log n)$ -time [11, 12]). Another shape for which polynomial-time algorithms exist is that of *tree* tensor networks [25, 23].

Another prominent way to optimize contraction orders is via the tree decomposition of the *line graph* representation of the tensor network [15, 5, 19]. In particular, this results in a contraction order with a maximal intermediate tensor rank equal to the treewidth of the tree decomposition. Loosely speaking, treewidth measures how tree-like a graph is. This does not directly solve our problem since finding the tree decompositions of the smallest treewidth is itself hard [14]. Two well-known frameworks to find good tree decompositions are **QuickBB** [6] and **FlowCutter** [24, 9].

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** [20] and **cotengra** [7], 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 [18], along with a Bayesian optimization approach, which has been later refined [21]. Other algorithms adopted from database query optimization are implemented in **netzwerk** [23]. Another option is to *learn* the best contraction orders, e.g., using reinforcement learning [16].

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 [22].

Worth mentioning is the SQL-view of einsum [2]: 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 [13], can be much faster than using the de-facto **numpy**-array implementation.

Chapter 12

Tensorgrad

Implementation details

12.1 Isomorphisms

There is actually a concept of “tensor isomorphism”, but it’s basically just the same as graph isomorphism.

We need to understand isomorphisms in many different parts of the code.

12.1.1 In Products

- Cancelling / combining equal parts of a product This is actually extra hard, because you have to collect a subset of nodes that constitute isomorphic subgraphs. Right now we hack this a bit by just considering separate components of the product.

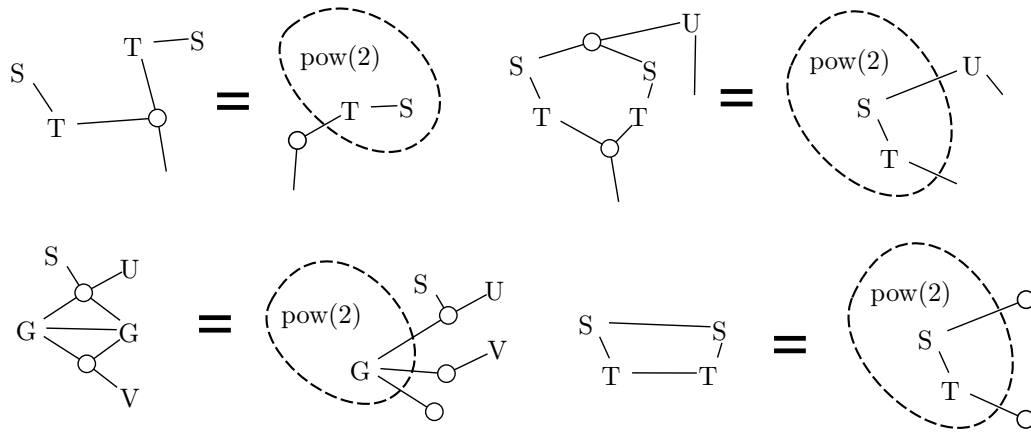


Figure 12.1: Combining equal parts of a product

Basically the problem is:

1. You are given a multigraph G with nodes V and edges E .
2. Nodes and edges are all labeled.
3. You are to find two disjoint subsets V_1 and V_2 of V such that the subgraphs G_1 and G_2 induced by V_1 and V_2 are isomorphic. Also, under the isomorphism, the labels of the nodes and edges in G_1 and G_2 are the same.

The problem is probably NP-hard, but it might still have an algorithm that's faster than 2^n trying all subsets. In particular, we might modify the VF2 algorithm, which iteratively tries to match nodes in G_1 and G_2 . The NetworkX library already has a GraphMatcher, which searches for isomorphic subgraphs. It might be extendable to our problem... But honestly I don't know if we even want to solve this problem in the most general, since it corresponds a bit to factoring the graph. And we don't do factoring, just as we don't do inverse distribution.

In either case, it's clear that we need to be able to compare nodes and edges for isomorphism.

Also, the basic usecase of isomorphism canonaization in products is simply to compute the canonical product itself from its parts. Part of our approach here is taking the outer edges and turning them into nodes, so they can be colored.

12.1.2 In Sums

When deciding whether $A + B$ is equal to $2A$ we need to check if A and B are isomorphic. But we also need to do this under the current renaming of the edges. That's why you can't just transform $A + A^T = 2A$.

The way it actually works in my code is

```

1 def key_fn(t: Tensor):
2     # Align tensor edges to have the same order, using Sum's order as
      reference.
3     canons = [t.canonical_edge_names[t.edges.index(e)] for e in self.edges]
4     return hash((t.canon,) + tuple(canons))
5
6 ws_tensors = TensorDict(key_fn=key_fn, default_fn=int)
7 for w, t in zip(weights, tensors):
8     ws_tensors[t] += w
9 ws_tensors = [(w, t) for t, w in ws_tensors.items()]

```

which says that I'm using for a hash, the canonical form of the tensor, plus the canonical form of the edges in the order of the edges in the sum. These are basically the orbits, meaning that if the summed tensor has a symmetry, we are allowed to "flip" it to make the summands isomorphic.

In the "compute canonical" method, we do more or less the same, but we also include the weights.

```

1 def _compute_canonical(self):
2     hashes = []
3     for e in self.edges:
4         canons = [t.canonical_edge_names[t.edges.index(e)] for t in self.
      tensors]
5         hashes.append(hash(("Sum",) + tuple(sorted(zip(self.weights, canons)
      )))))

```

```

6     base = hash(("Sum", len(self.tensors)))
7     hashes = [hash((base, h)) for h in hashes]
8     return base, hashes

```

In the future we want to use symmetry groups instead. What would be the symmetry group of a sum? It's the diagonal of the product of the symmetry groups of the summands. How can we find the generators of this group? Maybe we should just construct some joint graph and then find the automorphisms of that graph.

Alternatively we can use sympy. It is not known whether this problem is solvable in polynomial time. I think Babai proved that it is quasi-polynomial but not with a practical algorithm. Incidentally the problems of intersections of subgroups, centralizers of elements, and stabilizers of subsets of $\{1, \dots, n\}$ have been proved (by Eugene Luks) to be polynomially equivalent.

Actually making a graph and using nauty is a really good idea, since it would be able to detect that $A + A^T$ is symmetric. Just taking the intersection of the automorphism groups of the summands would not find that.

Another option is to convert the sum to a function... But no, that's weird. That would require me to support functions with arbitrary numbers of inputs, which is not currently the case.

12.1.3 In Evaluation

When evaluating a tensor, we can look at the graph of the tensor and see if it's isomorphic to a previously evaluated tensor. This is an example where we don't really need a canonical form, but an approximate hash plus vf2 would be fine. Also note that in this case we don't care about the edge renaming, because we can just rename the edges before we return the tensor. E.g. if we have already evaluated A , we can use that to get A^T easily.

12.1.4 In Variables

In variables we include the name of the variable in the hash. Basically we assume that variables named the same refer to the same data.

```

1     base = hash(("Variable", self.name))
2     return base, [hash((base, e)) for e in self.original_edges]

```

For the original canonical edge names, we use the edge names before renaming. This means, in the case of A^T that will have the same hash as A . But because it's renamed, the `t.index` call in the Sum will flip the edges.

We could imagine variables taking an automorphism group as an argument, which would allow us to define variables with different symmetries. Such as a symmetric matrix A where $A + A^T$ is actually $2A$.

12.1.5 In Constants

When computing the canonical form of a constant, like `Zero` or `Copy` we don't care about the edge names. I guess because the constants we use are all maximally symmetric? We currently include the constants `tag`, which is the hash of the variable that it came from, if any.

12.1.6 In Functions

One issue is that while the original names are usually part of the function definition, the new edges added by differentiation are often automatically generated based on the context, so they shouldn't really be part of the canonical form.

In contrast to Sum, we don't sort the canons here, since the order of the inputs matters.

Maybe functions should be allowed to transform the symmetry group? E.g. if we have a function that takes a symmetric matrix and returns a symmetric matrix, we should be able to use the symmetry group of the input to simplify the output.

12.1.7 In Derivatives

All we do is hashing the tensor and the wrt. And then add new edges for the derivative.

12.1.8 Other

For some tensors there might be edge dimension relations that aren't equivalences. For example, a flatten tensor would have the "out" edge dimension equal to the product of the "in" edge dimensions.

In a previous version I had every tensor register a "callback" function. Whenever an edge dimension "became available", the tensor would get a chance to emit new edge dimensions. However, this was a lot more work for each tensor to implement, and not needed for any of the existing tensors.

12.2 Renaming

This is an important part of the code.

12.3 Evaluation

An important part of evaluation is determining the dimension of each edge. To do this, I'm basically creating a full graph of the tensor, using a function called `edge_equivalences` which a list of tuples $((t_1, e_1), (t_2, e_2))$, indicating that edge e_1 of tensor t_1 is equivalent to edge e_2 of tensor t_2 . Note that the same edge name can appear multiple times in the graph, so we need to keep track of the tensor as well.

For variables, since the user gives edge dimensions in terms of variables, it's important to keep track of renamed edge names:

```
1 for e1, e2 in zip(self.original_edges, self.edges):
2     yield (self, e1), (self, e2)
```

For constants, there might be some equivalences based on tensors that the constant was derived from.

```
1 def edge_equivalences(self):
2     if self.link is not None:
3         yield from self.link.edge_equivalences()
4         for e in self.link.edges:
```

```

5         if e in self.edges:
6             yield (self, e), (self.link, e)

```

For the copy tensor, everything is equivalent:

```

1 def edge_equivalences(self):
2     yield from super().edge_equivalences()
3     for e in self.edges[1:]:
4         yield (self, self.edges[0]), (self, e)

```

For functions we can't really say anything about the edges of the function itself (`self.edges_out`), but at least we can say something about the broadcasted edges.

```

1 for t, *inner_edges in self.inputs:
2     yield from t.edge_equivalences()
3     for e in t.edges:
4         if e not in inner_edges:
5             yield (t, e), (self, e)

```

We could maybe also say that input edges with the same name are equivalent?

For products, we look at each edge (t_1, e, t_2) and yield $(t_1, e), (t_2, e)$. However for the free edges, (t, e) , we match them with ourselves, $(t, e), (self, e)$.

```

1 def edge_equivalences(self):
2     pairs = defaultdict(list)
3     for t in self.tensors:
4         yield from t.edge_equivalences()
5         for e in t.edges:
6             pairs[e].append(t)
7     for e, ts in pairs.items():
8         if len(ts) == 1:
9             yield (self, e), (ts[0], e)
10        else:
11            t1, t2 = ts
12            yield (t1, e), (t2, e)

```

Similarly, for sums, everything is just matched with ourselves:

```

1 def edge_equivalences(self):
2     for t in self.tensors:
3         yield from t.edge_equivalences()
4         for e in t.edges:
5             yield (t, e), (self, e)

```

Finally, we use BFS to propagate the edge dimensions from the variables (which are given by the user) to the rest of the graph.

Why is it even necessary for non-variables to know the edge dimensions? Mostly because of copy tensors, which we use for hyper edges, and have to construct. Could we get rid of this if we computed hyper-edges more efficiently without copy's? There are also sometimes "detached" copies...

Also, an alternative idea would be to actually construct the full graph. I originally didn't think this would be possible because of the Sum's which aren't really graphs. But maybe with the new approach of using nauty, we could actually do this.

12.3.1 Products

We simply evaluate the tensors in the product and give them to einsum.

12.4 Simplification Rules

There are a bunch of these.

Mostly we can do everything in a single depth-first pass, but a few times we need to do multiple passes. That can be done with the full-simplify method, which repeatedly calls simplify until nothing changes.

Bibliography

- [1] John Aldrich. *The Origins of Mathematical Words: A Comprehensive Dictionary of Latin, Greek, and Arabic Roots*. Mathematical Association of America, 2010.
- [2] 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.
- [3] 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.
- [4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
- [5] 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.
- [6] 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.
- [7] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, 2021.
- [8] greg (<https://math.stackexchange.com/users/357854/greg>). Proving that $\text{vec}(a \text{diag}(b) c) = ((c^t \otimes 1_a) \odot (1_c \otimes a)) b$. Mathematics Stack Exchange. URL:<https://math.stackexchange.com/q/2993406> (version: 2018-11-11).
- [9] Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *Journal of Experimental Algorithmics (JEA)*, 23:1–34, 2018.
- [10] William Rowan Hamilton. *Lectures on Quaternions*. Hodges and Smith, 1853.
- [11] T. C. Hu and M. T. Shing. Computation of matrix chain products. part I. *SIAM J. Comput.*, 11(2):362–373, 1982.
- [12] T. C. Hu and M. T. Shing. Computation of matrix chain products. part II. *SIAM J. Comput.*, 13(2):228–251, 1984.

- [13] 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.
- [14] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. *SIAM Journal on Computing*, (0):FOCS21–174, 2023.
- [15] Igor L Markov and Yaoyun Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, 2008.
- [16] 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.
- [17] Robert N. C. Pfeifer, Jutho Haegeman, and Frank Verstraete. Faster identification of optimal contraction sequences for tensor networks. *Phys. Rev. E*, 90:033315, 2014.
- [18] 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.
- [19] Roman Schutschi, Danil Lykov, and Ivan Oseledets. Adaptive algorithm for quantum circuit simulation. *Phys. Rev. A*, 101:042335, Apr 2020.
- [20] 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.
- [21] 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.
- [22] Mihail Stoian and Andreas Kipf. Dpconv: Super-polynomially faster join ordering, 2024.
- [23] 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.
- [24] Ben Strasser. Computing tree decompositions with flowcutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017.
- [25] 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.
- [26] 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 13

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.