Named Tensor Notation

Anonymous authors
Paper under double-blind review

Abstract

We propose a notation for tensors with named axes, which relieves the author, reader, and future implementers of machine learning models from the burden of keeping track of the order of axes and the purpose of each. The notation makes it easy to lift operations on low-order tensors to higher order ones, for example, from images to minibatches of images, or from an attention mechanism to multiple attention heads.

After a brief overview and formal definition of the notation, we illustrate it through several examples from modern machine learning, from building blocks like attention and convolution to full models like Transformers and LeNet. We then discuss differential calculus in our notation and compare with some alternative notations. Our proposals build on ideas from many previous papers and software libraries. We hope that this document will encourage more authors to use named tensors, resulting in clearer papers and more precise implementations.

1 Introduction

Formal descriptions of neural networks primarily adopt the notation of vectors and matrices from applied linear algebra (Goodfellow et al., 2016). When used to describe vector spaces, this notation is both concise and unambiguous. However, when applied to neural networks, these properties are lost. Consider the equation for attention as notated in the Transformer paper (Vaswani et al., 2017):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

The equation relates Q, K, and V (for query, key, and value, respectively) as sequences of feature vectors, packed into possibly identically-sized matrices. While concise, even this short equation is ambiguous. Does the product $QK^\top$ sum over the sequence, or over the features? We know that it sums over columns, but there is not enough information to know what the columns represent. Is the softmax taken over the query sequence or the key sequence? The usual notation does not offer an answer. Perniciously, the implementation of an incorrect interpretation might still run without errors. With the addition of more axes, like multiple attention heads or multiple sentences in a minibatch, the notation becomes even more cumbersome.

We propose an alternative mathematical notation for tensors with *named axes*.[1] The notation has a formal underpinning, but is hopefully intuitive enough that machine learning researchers can understand it without much effort. In named tensor notation, the above equation becomes

$$\text{Attention}\colon \mathbb{R}^{\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}\times\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}\times\mathsf{val}} \to \mathbb{R}^{\mathsf{val}}$$

$$\text{Attention}(Q, K, V) = \underset{\mathsf{seq}}{\text{softmax}}\left(\frac{Q \underset{\mathsf{key}}{\odot} K}{\sqrt{|\mathsf{key}|}}\right) \underset{\mathsf{seq}}{\odot} V.$$

The type signature introduces three named axes: the key axis is for features of queries and keys, the val axis is for features of values, and the seq axis is for tokens in a sequence. This notation makes the types of each input tensor explicit. Tensor Q is a query vector that is compared with key vectors, so it has a key axis.

[1] We follow NumPy in using the term *axis*. Other possible terms would be *index*, *dimension*, *way*, or *mode* (Tucker, 1964), but we felt that *axis* had the least potential for confusion.

Tensor K is a sequence of key vectors, so it has seq and key axes. Tensor V is a sequence of value vectors, so it has seq and val axes. Unlike with matrix notation, the reader is not required to remember whether seq corresponds to rows or columns in either of these tensors.

The function itself uses the named axes to precisely apply operations. The expression $Q \underset{\mathsf{key}}{\odot} K$ is a dot product over the key axis shared between K and Q; there is no ambiguity about rows or columns. Similarly, the softmax function is annotated with the axis along which it is applied, removing any ambiguity or reliance on convention.

Furthermore, named tensor notation naturally extends to *lifting* (also known as vectorizing and/or broadcasting) a function to tensors with more axes. For example, if instead of being a tensor with the single axis key, Q has three axes key, seq and batch (corresponding to tokens of a sequence and examples in a minibatch, respectively) then the Attention function works as written, acting on each example in a minibatch in parallel. Similarly, we can also add a heads axis to the inputs to get multiple attention heads. These additional axes are often elided in neural network papers, possibly avoiding notational complexity, but possibly also hiding critical model details.

Our contributions. This work proposes a *mathematical notation* for named tensors and a fully specified *semantic interpretation* for the notation. Through examples we demonstrate that this notation enables specifying machine learning models and operations in a succinct but yet precise manner. The need for named tensors has been recognized by several software packages, including xarray (Hoyer & Hamman, 2017), Nexus (Chen, 2017), tsalib (Sinha, 2018), NamedTensor (Rush, 2019), named tensors in PyTorch (Torch Contributors, 2019), axisarrays (Bauman, 2018), and Dex (Maclaurin et al., 2019). While our notation is inspired by these, our focus is on mathematical notation to be used in papers, rather than code. We hope that, if adopted by researchers in papers, this notation will inspire software packages as well and eventually lead to both clearer papers and more correct implementations.

2 Named Tensors

In standard notation, a vector, matrix, or tensor is indexed by an integer or sequence of integers; if it has dimensions $n_1, \ldots, n_r$, it can be thought of as a map that takes as input $(i_1, \ldots, i_r) \in [n_1] \times \cdots \times [n_r]$ and outputs a real number (or an element of a different field). For example, if $A \in \mathbb{R}^{3\times 3}$, then the order of the two axes matters: $A_{1,3}$ and $A_{3,1}$ are not the same element. It is up to the reader to remember what each axis of each tensor stands for. This problem is exacerbated in modern machine learning, where tensors have multiple axes with different meanings (batches, channels, etc.), and different operations act on different axes.

In contrast, in a *named tensor*, each axis has a *name* that describes it and ensures there is no confusion between axes. If it has axes $\mathsf{ax}_1[n_1], \ldots, \mathsf{ax}_r[n_r]$ (with $\mathsf{ax}_1, \ldots, \mathsf{ax}_r$ being distinct names), it can be thought of as a map that takes as input a *record* $\{\mathsf{ax}_1[i_1], \ldots, \mathsf{ax}_r[i_r]\}$, with $i_1 \in [n_1], \ldots, i_r \in [n_r]$, and outputs a field element. The key difference is that, rather than taking as input an ordered tuple of indices, a named tensor takes as input a *record*, which is an unordered set of named indices.

2.1 By example

For example, if A represents a 3×3 grayscale image, we can make it a named tensor like so (writing it two equivalent ways to show that the order of axes does not matter):

$$A \in \mathbb{R}^{\mathsf{height}[3]\times\mathsf{width}[3]} = \mathbb{R}^{\mathsf{width}[3]\times\mathsf{height}[3]}$$

$$A = \begin{matrix}\\ \mathsf{height}\end{matrix}\begin{matrix}\begin{matrix}\mathsf{width}\end{matrix}\\ \begin{bmatrix}3 & 1 & 4\\ 1 & 5 & 9\\ 2 & 6 & 5\end{bmatrix}\end{matrix} = \begin{matrix}\\ \mathsf{width}\end{matrix}\begin{matrix}\begin{matrix}\mathsf{height}\end{matrix}\\ \begin{bmatrix}3 & 1 & 2\\ 1 & 5 & 6\\ 4 & 9 & 5\end{bmatrix}\end{matrix}.$$

We access elements of A using named indices, whose order again does not matter: $A_{\mathsf{height}(1),\mathsf{width}(3)} = A_{\mathsf{width}(3),\mathsf{height}(1)} = 4$. We also allow partial indexing:

$$A_{\mathsf{height}(1)} = \overset{\mathsf{width}}{\begin{bmatrix}3 & 1 & 4\end{bmatrix}} \qquad A_{\mathsf{width}(3)} = \overset{\mathsf{height}}{\begin{bmatrix}4 & 9 & 5\end{bmatrix}}.$$

It does not matter if we write $A_{\mathsf{height}(1)}$ or $A_{\mathsf{width}(3)}$ as row and column vectors. In many contexts, an axis name is used with only one size. If so, we can simply write height for the unique axis with name height, as in $\mathbb{R}^{\mathsf{height}\times\mathsf{width}}$. We can leave the size of an axis unspecified at first, and specify its size later (e.g., deferring it to an appendix on experimental details). For example, we can specify $|\mathsf{height}| = |\mathsf{width}| = 28$ if we want to prescribe the precise size of an image, or just write $|\mathsf{height}| = |\mathsf{width}|$ to specify that it's a square image.

What are good choices for axis names? We recommend meaningful *words* instead of single letters, and we recommend words that describe a *whole* rather than its parts. For example, to represent a minibatch of examples, we would name the axis batch; to represent a sequence of tokens, we would name the axis seq. When no name for the whole is available, we recommend the plural form of the name of part. For example, if we wanted A to have red, green, and blue channels, we would name the axis chans. Please see §4 for more examples.

2.2 Formal definition

We now define formally the notation we use.

Definition 1 (Names, indices, and axes)**.** An *axis* is a pair, written $\mathsf{ax}[I]$, where

- ax is the *name* of the axis, which is simply a string of letters. We write both names and variables ranging over names using sans-serif font.
- I is a set of *indices*. In this paper, I is always of the form $\{1, \dots, n\}$ for some n, so we abbreviate $\mathsf{ax}[\{1, \dots, n\}]$ as $\mathsf{ax}[n]$.

In many contexts, there is only one axis with name ax, and so we refer to the axis simply as ax. The context always makes it clear whether ax is a name or an axis. If ax is an axis, we write $\operatorname{ind}(\mathsf{ax})$ for its index set, and we write $|\mathsf{ax}|$ as shorthand for $|\operatorname{ind}(\mathsf{ax})|$.

Definition 2 (Named indices and records)**.** If $\mathsf{ax}[I]$ is an axis and $i \in I$, then a *named index* is a pair, written $\mathsf{ax}(i)$. A *record* is a set of named indices $\{\mathsf{ax}_1(i_1), \dots, \mathsf{ax}_r(i_r)\}$, where $\mathsf{ax}_1, \dots \mathsf{ax}_r$ are pairwise distinct names.

Definition 3 (Shapes)**.** A *shape* is a set of axes, written $\mathsf{ax}_1[I_1] \times \cdots \times \mathsf{ax}_r[I_r]$, where $\mathsf{ax}_1, \dots \mathsf{ax}_r$ are pairwise distinct names. We write $\emptyset$ for the empty shape. A shape defines a set of records:

$$\operatorname{rec}(\mathsf{ax}_1[I_1] \times \cdots \times \mathsf{ax}_r[I_r]) = \{\{\mathsf{ax}_1(i_1), \dots, \mathsf{ax}_r(i_r)\} \mid i_1 \in I_1, \dots, i_r \in I_r\}.$$

We say two shapes $\mathcal{S}$ and $\mathcal{T}$ are *compatible* if whenever $\mathsf{ax}[I] \in \mathcal{S}$ and $\mathsf{ax}[J] \in \mathcal{T}$, then $I = J$. We say that $\mathcal{S}$ and $\mathcal{T}$ are *orthogonal* if there is no ax such that $\mathsf{ax}[I] \in \mathcal{S}$ and $\mathsf{ax}[J] \in \mathcal{T}$ for any I, J. If $t \in \operatorname{rec}\mathcal{T}$ and $\mathcal{S} \subseteq \mathcal{T}$, then we write $t|_{\mathcal{S}}$ for the unique record in $\operatorname{rec}\mathcal{S}$ such that $t|_{\mathcal{S}} \subseteq t$.

Definition 4 (Named tensors)**.** Let F be a field and let $\mathcal{S}$ be a shape. Then a *named tensor over F with shape $\mathcal{S}$* is a mapping from $\operatorname{rec}\mathcal{S}$ to F. We write the set of all named tensors with shape $\mathcal{S}$ as $F^{\mathcal{S}}$.

We don't make any distinction between a scalar (an element of F) and a named tensor with empty shape (an element of $F^{\emptyset}$).

If $A \in F^{\mathcal{S}}$, then we access an element of A by applying it to a record $s \in \operatorname{rec}\mathcal{S}$; but we write this using the usual subscript notation: A_s rather than $A(s)$. To avoid clutter, in place of $A_{\{\mathsf{ax}_1(i_1),\dots,\mathsf{ax}_r(i_r)\}}$, we usually write $A_{\mathsf{ax}_1(i_1),\dots,\mathsf{ax}_r(x_r)}$. When a named tensor is an expression like $(A + B)$, we index it by surrounding it with square brackets like this: $[A + B]_{\mathsf{ax}_1(i_1),\dots,\mathsf{ax}_r(x_r)}$.

We also allow partial indexing. If A is a tensor with shape $\mathcal{T}$ and $s \in \operatorname{rec}\mathcal{S}$ where $\mathcal{S} \subseteq \mathcal{T}$, then we define A_s to be the named tensor with shape $\mathcal{T} \setminus \mathcal{S}$ such that, for any $t \in \operatorname{rec}(\mathcal{T} \setminus \mathcal{S})$,

$$[A_s]_t = A_{s \cup t}.$$

(For the edge case $\mathcal{T} = \emptyset$, our definitions for indexing and partial indexing coincide: one gives a scalar and the other gives a tensor with empty shape, but we don't distinguish between the two.)

3 Operations

A significant benefit of named tensor notation is that it allows one to unambiguously specify *operations* that map tensors to tensors, and defines precisely how operations can be *lifted* when an operation is applied to tensors with more axes than are present in its signature and how *broadcasting* happens when different arguments add different axes.

We start with the formal definition of named tensor operations, then show how this definition leads to many common operations.

3.1 Formal definition

Definition 5 (Named tensor operation)**.** Let $\mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T}$ be shapes. A *named tensor operation* with signature $F^{\mathcal{S}_1} \times \cdots \times F^{\mathcal{S}_k} \to F^{\mathcal{T}}$ is a function $f : F^{\mathcal{S}_1} \times \cdots \times F^{\mathcal{S}_k} \to F^{\mathcal{T}}$ that takes as input tensors $A_1, \ldots, A_k$ of shapes $\mathcal{S}_1, \ldots, \mathcal{S}_k$ respectively and maps it into a tensor B of shape $\mathcal{T}$.

We can now define how to lift such operations to higher-order tensors.

Definition 6 (lifting, unary)**.** Let $f\colon F^{\mathcal{S}} \to G^{\mathcal{T}}$ be a function from tensors to tensors. For any shape $\mathcal{S}'$ orthogonal to both $\mathcal{S}$ and $\mathcal{T}$, we can define the *lift* $f^{\mathcal{S}'}$ of f with the shape $\mathcal{S}'$ to be the map

$$\begin{aligned} f^{\mathcal{S}'} &: F^{\mathcal{S} \cup \mathcal{S}'} \to G^{\mathcal{T} \cup \mathcal{S}'} \\ \left[f^{\mathcal{S}'}(A)\right]_s &= f(A_s) \qquad \text{for all } A \in F^{\mathcal{S} \cup \mathcal{S}'} \text{ and } s \in \operatorname{rec} \mathcal{S}'. \end{aligned}$$

Usually, we simply write f instead of $f^{\mathcal{S}'}$. That is, for every tensor A with shape $\mathcal{R} \supseteq \mathcal{S}$, we let $f(A) = f^{\mathcal{R} \setminus \mathcal{S}}(A)$.

If f is a multary function, we can lift each of its arguments to larger shapes, and we don't have to add the same axes to all the arguments; an axis present in one argument but not another is *broadcast* from the former to the latter. We consider just the case of two arguments; three or more arguments are analogous.

Definition 7 (lifting, binary)**.** Let $f\colon F^{\mathcal{S}} \times G^{\mathcal{T}} \to H^{\mathcal{U}}$ be a binary function from tensors to tensors. For any shapes $\mathcal{S}'$ and $\mathcal{T}'$ that are compatible with each other and orthogonal to $\mathcal{S}$ and $\mathcal{T}$, respectively, and $\mathcal{S}' \cup \mathcal{T}'$ is orthogonal to $\mathcal{U}$, we can lift f to:

$$\begin{aligned} f^{\mathcal{S}' \times \mathcal{T}'} &: F^{\mathcal{S} \cup \mathcal{S}'} \times G^{\mathcal{T} \cup \mathcal{T}'} \to H^{\mathcal{U} \cup \mathcal{S}' \cup \mathcal{T}'} \\ \left[f^{\mathcal{S}' \times \mathcal{T}'}(A, B)\right]_s &= f\left(A_{s|_{\mathcal{S}'}}, B_{s|_{\mathcal{T}'}}\right) \qquad \text{for all } A \in F^{\mathcal{S} \cup \mathcal{S}'},\ B \in F^{\mathcal{T} \cup \mathcal{T}'},\ s \in \operatorname{rec}(\mathcal{S}' \cup \mathcal{T}'). \end{aligned}$$

Again, we usually write f instead of $f^{\mathcal{S}' \times \mathcal{T}'}$.

3.2 Elementwise operations and broadcasting

Any function from a scalar to a scalar can be applied elementwise to a named tensor. For example, given the logistic sigmoid function,

$$\begin{aligned} \sigma &\colon \mathbb{R} \to \mathbb{R} \\ \sigma(x) &= \frac{1}{1 + \exp(-x)} \end{aligned}$$

we can lift it to tensors:

$$\sigma(A) = \begin{array}{c} \\ \mathsf{height} \end{array} \overset{\mathsf{width}}{\begin{bmatrix} \frac{1}{1+\exp(-3)} & \frac{1}{1+\exp(-1)} & \frac{1}{1+\exp(-4)} \\ \frac{1}{1+\exp(-1)} & \frac{1}{1+\exp(-5)} & \frac{1}{1+\exp(-9)} \\ \frac{1}{1+\exp(-2)} & \frac{1}{1+\exp(-6)} & \frac{1}{1+\exp(-5)} \end{bmatrix}}.$$

Similarly for rectified linear units ($\text{relu}(x) = \max(0, x)$), negation, and so on.

Similarly, any function $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$, like addition ($+$), real multiplication (which we write as $\odot$), and so on, can be applied to two named tensors with the same shape. But if we apply a binary function or operator to tensors with different shapes, then broadcasting applies. For example, let

$$x \in \mathbb{R}^{\mathsf{height}[3]} \qquad\qquad y \in \mathbb{R}^{\mathsf{width}[3]}$$

$$x = \begin{matrix}\mathsf{height}\end{matrix}\begin{bmatrix}2\\7\\1\end{bmatrix} \qquad\qquad y = \begin{matrix}\\\\\mathsf{width}\end{matrix}\begin{bmatrix}1 & 4 & 1\end{bmatrix}.$$

(We write x as a column just to make the broadcasting easier to visualize.) Then, to evaluate $A + x$, we effectively replace x with a new tensor with a copy of x for every index of axis width. Likewise for $A + y$:

$$A + x = \begin{matrix}\\\mathsf{height}\end{matrix}\begin{matrix}\begin{matrix}\mathsf{width}\end{matrix}\\\begin{bmatrix}3+2 & 1+2 & 4+2\\1+7 & 5+7 & 9+7\\2+1 & 6+1 & 5+1\end{bmatrix}\end{matrix} \qquad A + y = \begin{matrix}\\\mathsf{height}\end{matrix}\begin{matrix}\begin{matrix}\mathsf{width}\end{matrix}\\\begin{bmatrix}3+1 & 1+4 & 4+1\\1+1 & 5+4 & 9+1\\2+1 & 6+4 & 5+1\end{bmatrix}\end{matrix}.$$

3.3 Reductions

The same rules apply to functions from vectors to scalars, called *reductions*. We always specify which axis a reduction applies to using a subscript (equivalent to the `axis` argument in NumPy and `dim` in PyTorch). For example, we can define summation:

$$\sum_{\mathsf{ax}} - \colon \mathbb{R}^{\mathsf{ax}[I]} \to \mathbb{R}$$

$$\sum_{\mathsf{ax}} X = \sum_{i \in I} X_{\mathsf{ax}(i)}$$

and use it on A from above:

$$\sum_{\mathsf{height}} A = \sum_i A_{\mathsf{height}(i)} = \begin{matrix}\\\mathsf{width}\end{matrix}\begin{bmatrix}3+1+2 & 1+5+6 & 4+9+5\end{bmatrix}$$

$$\sum_{\mathsf{width}} A = \sum_j A_{\mathsf{width}(j)} = \begin{matrix}\\\mathsf{height}\end{matrix}\begin{bmatrix}3+1+4\\1+5+9\\2+6+5\end{bmatrix}.$$

We can also write multiple names to sum over multiple axes:

$$\sum_{\substack{\mathsf{height}\\\mathsf{width}}} A = \sum_i \sum_j A_{\mathsf{height}(i),\mathsf{width}(j)} = 3+1+4+1+5+9+2+6+5.$$

But a summation with an index variable (like i or j above) is a standard summation over values of that variable, and a summation with no subscript is a standard summation over a set.

Other examples of reductions include:

$$\operatorname*{norm}_{\mathsf{ax}} X = \sqrt{\sum_{\mathsf{ax}} X^2}$$

$$\operatorname*{min}_{\mathsf{ax}} X = \min\{X_{\mathsf{ax}(i)} \mid i \in I\} \qquad\qquad \operatorname*{max}_{\mathsf{ax}} X = \max\{X_{\mathsf{ax}(i)} \mid i \in I\}$$

$$\operatorname*{mean}_{\mathsf{ax}} X = \frac{1}{|\mathsf{ax}|} \sum_{\mathsf{ax}} X \qquad\qquad \operatorname*{var}_{\mathsf{ax}} X = \frac{1}{|\mathsf{ax}|} \sum_{\mathsf{ax}} (X - \operatorname*{mean}_{\mathsf{ax}} X)^2$$

The vector dot-product is a function from *two* vectors to a scalar. We write it as follows:

$$- \underset{\mathsf{ax}}{\odot} - \colon \mathbb{R}^{\mathsf{ax}[I]} \times \mathbb{R}^{\mathsf{ax}[I]} \to \mathbb{R}$$

$$X \underset{\mathsf{ax}}{\odot} Y = \sum_{i \in I} X_{\mathsf{ax}(i)} Y_{\mathsf{ax}(i)}$$

The dot-product generalizes to named tensors to give the ubiquitous *contraction* operator, which can be thought of as elementwise multiplication followed by summation over one axis:

$$A \underset{\mathsf{height}}{\odot} x = \sum_i A_{\mathsf{height}(i)} x_{\mathsf{height}(i)} = \begin{bmatrix} 3 \cdot 2 + 1 \cdot 7 + 2 \cdot 1 & 1 \cdot 2 + 5 \cdot 7 + 6 \cdot 1 & 4 \cdot 2 + 9 \cdot 7 + 5 \cdot 1 \end{bmatrix}_{\mathsf{width}}$$

$$A \underset{\mathsf{width}}{\odot} y = \sum_j A_{\mathsf{width}(j)}\, y_{\mathsf{width}(j)} = \mathsf{height} \begin{bmatrix} 3 \cdot 1 + 1 \cdot 4 + 4 \cdot 1 \\ 1 \cdot 1 + 5 \cdot 4 + 9 \cdot 1 \\ 2 \cdot 1 + 6 \cdot 4 + 5 \cdot 1 \end{bmatrix}.$$

Again, we can write multiple names to contract multiple axes at once. A $\odot$ with no axis name under it contracts zero axes and is equivalent to elementwise multiplication, so we use $\odot$ for elementwise multiplication as well. The contraction operator can be used for many multiplication-like operations:

$$x \underset{\mathsf{height}}{\odot} x = \sum_i x_{\mathsf{height}(i)}\, x_{\mathsf{height}(i)} \qquad \text{inner product}$$

$$[x \odot y]_{\mathsf{height}(i), \mathsf{width}(j)} = x_{\mathsf{height}(i)}\, y_{\mathsf{width}(j)} \qquad \text{outer product}$$

$$A \underset{\mathsf{width}}{\odot} y = \sum_i A_{\mathsf{width}(i)}\, y_{\mathsf{width}(i)} \qquad \text{matrix-vector product}$$

$$x \underset{\mathsf{height}}{\odot} A = \sum_i x_{\mathsf{height}(i)}\, A_{\mathsf{height}(i)} \qquad \text{vector-matrix product}$$

$$A \underset{\mathsf{width}}{\odot} B = \sum_i A_{\mathsf{width}(i)} \odot B_{\mathsf{width}(i)} \qquad \text{matrix-matrix product } (B \in \mathbb{R}^{\mathsf{width} \times \mathsf{width}'})$$

3.4 Vectors to vectors

Functions from vectors to vectors ($\mathbb{R}^{\mathsf{ax}[I]} \to \mathbb{R}^{\mathsf{ax}[I]}$) are particularly problematic in standard notation, which does not provide any way (to our knowledge) of specifying which axis the operation should be performed over. Such functions include:

$$\underset{\mathsf{ax}}{\operatorname{softmax}} X = \frac{\exp A}{\sum_{\mathsf{ax}} \exp X}$$

$$\underset{\mathsf{ax}}{\operatorname{argmax}} X = \lim_{\alpha \to \infty} \underset{\mathsf{ax}}{\operatorname{softmax}}\, \alpha X$$

$$\underset{\mathsf{ax}}{\operatorname{argmin}} X = \lim_{\alpha \to -\infty} \underset{\mathsf{ax}}{\operatorname{softmax}}\, \alpha X$$

For example, we can clearly distinguish between two ways of performing a softmax on A:

$$\underset{\mathsf{height}}{\operatorname{softmax}} A = \mathsf{height} \begin{bmatrix} \frac{\exp 3}{\exp 3 + \exp 1 + \exp 2} & \frac{\exp 1}{\exp 1 + \exp 5 + \exp 6} & \frac{\exp 4}{\exp 4 + \exp 9 + \exp 5} \\ \frac{\exp 1}{\exp 3 + \exp 1 + \exp 2} & \frac{\exp 5}{\exp 1 + \exp 5 + \exp 6} & \frac{\exp 9}{\exp 4 + \exp 9 + \exp 5} \\ \frac{\exp 2}{\exp 3 + \exp 1 + \exp 2} & \frac{\exp 6}{\exp 1 + \exp 5 + \exp 6} & \frac{\exp 5}{\exp 4 + \exp 9 + \exp 5} \end{bmatrix}_{\mathsf{width}}$$

$$\underset{\mathsf{width}}{\operatorname{softmax}} A = \mathsf{height} \begin{bmatrix} \frac{\exp 3}{\exp 3 + \exp 1 + \exp 4} & \frac{\exp 1}{\exp 3 + \exp 1 + \exp 4} & \frac{\exp 4}{\exp 3 + \exp 1 + \exp 4} \\ \frac{\exp 1}{\exp 1 + \exp 5 + \exp 9} & \frac{\exp 5}{\exp 1 + \exp 5 + \exp 9} & \frac{\exp 9}{\exp 1 + \exp 5 + \exp 9} \\ \frac{\exp 2}{\exp 2 + \exp 6 + \exp 5} & \frac{\exp 6}{\exp 2 + \exp 6 + \exp 5} & \frac{\exp 5}{\exp 2 + \exp 6 + \exp 5} \end{bmatrix}_{\mathsf{width}}$$

3.5 Renaming and reshaping

It's often useful to rename an axis (analogous to a transpose operation in standard notation). We can accomplish this using a function from vectors to vectors, but with different input and output axes:

$$[-]_{\mathsf{ax}\to\mathsf{ax}'} : \mathbb{R}^{\mathsf{ax}[I]} \to \mathbb{R}^{\mathsf{ax}'[I]}$$
$$[X_{\mathsf{ax}\to\mathsf{ax}'}]_{\mathsf{ax}'(i)} = X_{\mathsf{ax}(i)}$$

For example,

$$A_{\mathsf{height}\to\mathsf{height}'} = \begin{matrix}\\ \mathsf{height}'\end{matrix}\overset{\mathsf{width}}{\begin{bmatrix}3 & 1 & 4\\ 1 & 5 & 9\\ 2 & 6 & 5\end{bmatrix}}.$$

We can also define notation for reshaping two or more axes into one axis:

$$A_{(\mathsf{height},\mathsf{width})\to\mathsf{layer}} = \overset{\mathsf{layer}}{\begin{bmatrix}3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 & 5\end{bmatrix}}$$

The order of elements in the new axis is undefined. Authors who need a particular ordering may write a more specific definition.

4 Worked Examples: Neural Networks

In this section we give a series of worked examples illustrating how standard neural network model components can be written using named tensors. Appendix A builds some of these components into complete specifications of the Transformer and LeNet.

4.1 Feedforward neural networks

A multi-layer, feedforward neural network with different-sized layers can be written as:

$$X^0 \in \mathbb{R}^{\mathsf{input}}$$

$$X^1 = \sigma(W^1 \underset{\mathsf{input}}{\odot} X^0 + b^1) \qquad W^1 \in \mathbb{R}^{\mathsf{hidden}_1 \times \mathsf{input}} \qquad b^1 \in \mathbb{R}^{\mathsf{hidden}_1}$$

$$X^2 = \sigma(W^2 \underset{\mathsf{hidden}_1}{\odot} X^1 + b^2) \qquad W^2 \in \mathbb{R}^{\mathsf{hidden}_2 \times \mathsf{hidden}_1} \qquad b^2 \in \mathbb{R}^{\mathsf{hidden}_2}$$

$$X^3 = \sigma(W^3 \underset{\mathsf{hidden}_2}{\odot} X^2 + b^3) \qquad W^3 \in \mathbb{R}^{\mathsf{out} \times \mathsf{hidden}_2} \qquad b^3 \in \mathbb{R}^{\mathsf{out}}$$

The layer sizes can be specified by writing $|\mathsf{hidden}_1| = n_1$, etc. As noted above, σ is applied elementwise and does not require additional annotation.

Alternatively, the layer equation can be abstracted by writing:

$$\mathrm{FullConn}^l(x; W^l, b^l) = \sigma\left(W^l \underset{\mathsf{layer}}{\odot} x + b^l\right)_{\mathsf{layer}'\to\mathsf{layer}}$$

where

$$W^l \in \mathbb{R}^{\mathsf{layer}'[n_l] \times \mathsf{layer}[n_{l-1}]}$$
$$b^l \in \mathbb{R}^{\mathsf{layer}'[n_l]}.$$

We assume $\mathrm{FullConn}^l$ encapsulates both the equation for layer l as well as its parameters W^l, b^l (analogous to what TensorFlow and PyTorch *modules*). Since we chose to use the same axis name layer for all the layers (with different sizes n_l), $\mathrm{FullConn}^l$ temporarily computes its output over axis layer', then renames it back to layer. The network can be defined like this:

$$X^0 \in \mathbb{R}^{\mathsf{layer}[n_0]}$$

$$X^1 = \text{FullConn}^1(X^0)$$
$$X^2 = \text{FullConn}^2(X^1)$$
$$X^3 = \text{FullConn}^3(X^2).$$

4.2 Recurrent neural networks

As a second example, we consider a simple (Elman) RNN. This model is similar to the feedforward network, except that the number of timesteps is variable and parameters are shared over time. At each time step, it produces a tensor with a new axis hidden' which is then renamed hidden for the next step in the recursion.

$$\begin{aligned}
x^t &\in \mathbb{R}^{\mathsf{input}} && t = 1, \ldots, n \\
W^{\text{h}} &\in \mathbb{R}^{\mathsf{hidden} \times \mathsf{hidden}'} && |\mathsf{hidden}| = |\mathsf{hidden}'| \\
W^{\text{i}} &\in \mathbb{R}^{\mathsf{input} \times \mathsf{hidden}'} \\
b &\in \mathbb{R}^{\mathsf{hidden}'} \\
h^0 &\in \mathbb{R}^{\mathsf{hidden}} \\
h^t &= \sigma\left(W^{\text{h}} \underset{\mathsf{hidden}}{\odot} h^{t-1} + W^{\text{i}} \underset{\mathsf{input}}{\odot} x^t + b\right)_{\mathsf{hidden}' \to \mathsf{hidden}} && t = 1, \ldots, n
\end{aligned}$$

4.3 Attention

In the introduction (§1), we describe difficulties in interpreting the equation for attention as used with Transformers (Vaswani et al., 2017). In our notation, it looks like this:

$$\text{Attention}\colon \mathbb{R}^{\mathsf{key}} \times \mathbb{R}^{\mathsf{seq} \times \mathsf{key}} \times \mathbb{R}^{\mathsf{seq} \times \mathsf{val}} \to \mathbb{R}^{\mathsf{val}}$$
$$\text{Attention}(Q, K, V) = \underset{\mathsf{seq}}{\text{softmax}}\left(\frac{Q \underset{\mathsf{key}}{\odot} K}{\sqrt{|\mathsf{key}|}}\right) \underset{\mathsf{seq}}{\odot} V.$$

This definition takes a single query Q vector returns a single result vector (and actually could be further reduced to a scalar values as val is not strictly necessary). To apply to a sequence, we can give Q a seq' axis, and the function will compute an output sequence. Providing Q, K, and V with a heads axis lifts the function to compute multiple attention heads.

For Transformers we often need to apply a mask to ensure attention is only applied to valid keys (e.g. for causal language models). We can modify the equation to include this mask:

$$\text{Attention}\colon \mathbb{R}^{\mathsf{key}} \times \mathbb{R}^{\mathsf{seq} \times \mathsf{key}} \times \mathbb{R}^{\mathsf{seq} \times \mathsf{val}} \times \mathbb{R}^{\mathsf{seq}} \to \mathbb{R}^{\mathsf{val}}$$
$$\text{Attention}(Q, K, V, M) = \underset{\mathsf{seq}}{\text{softmax}}\left(\frac{Q \underset{\mathsf{key}}{\odot} K}{\sqrt{|\mathsf{key}|}} + M\right) \underset{\mathsf{seq}}{\odot} V.$$

Appendix A.1 includes a full specification of the complete Transformer model using the named tensor notation.

4.4 Convolution

Standard neural network convolutions can be specified by "unrolling" a vector and then applying a standard dot product. We define an axis-parameterized unrolling function that converts a one-axis tensor to a sequence of kernel sized vectors:

$$\underset{\substack{\mathsf{seq} \\ \mathsf{kernel}}}{\text{unroll}}\colon \mathbb{R}^{\mathsf{seq}[n]} \to \mathbb{R}^{\mathsf{seq}[n - |\mathsf{kernel}| + 1], \mathsf{kernel}}$$

$$\underset{\substack{\mathsf{seq}\\\mathsf{kernel}}}{\operatorname{unroll}} X = Y, \text{ where}$$
$$Y_{\mathsf{seq}(i),\mathsf{kernel}(j)} = X_{\mathsf{seq}(i+j-1)}.$$

A 1d convolution with input channels chans and output channels chans' consists of unrolling along the seq axis and then taking a dot product:

$$\begin{aligned}\text{Conv1d}\colon \mathbb{R}^{\mathsf{chans}\times\mathsf{seq}[n]} &\to \mathbb{R}^{\mathsf{chans}'\times\mathsf{seq}[n']}\\ \text{Conv1d}(X; W, b) &= W \underset{\substack{\mathsf{chans}\\\mathsf{kernel}}}{\odot} \underset{\substack{\mathsf{seq}\\\mathsf{kernel}}}{\operatorname{unroll}} X + b\end{aligned}$$

where

$$\begin{aligned}W &\in \mathbb{R}^{\mathsf{chans}'\times\mathsf{chans}\times\mathsf{kernel}}\\ b &\in \mathbb{R}\end{aligned}$$

Unrolling easily generalizes to higher-dimensional convolutions:

$$\begin{aligned}\text{Conv2d}\colon \mathbb{R}^{\mathsf{chans}\times\mathsf{height}[h]\times\mathsf{width}[w]} &\to \mathbb{R}^{\mathsf{chans}'\times\mathsf{height}[h']\times\mathsf{width}[w']}\\ \text{Conv2d}(X; W, b) &= W \underset{\substack{\mathsf{chans}\\\mathsf{kh,kw}}}{\odot} \underset{\substack{\mathsf{height}\\\mathsf{kh}}}{\operatorname{unroll}} \underset{\substack{\mathsf{width}\\\mathsf{kw}}}{\operatorname{unroll}} X + b\end{aligned}$$

where

$$\begin{aligned}W &\in \mathbb{R}^{\mathsf{chans}'\times\mathsf{chans}\times\mathsf{kh}\times\mathsf{kw}}\\ b &\in \mathbb{R}.\end{aligned}$$

4.5 Pooling

Pooling is similar to convolutions. We first define a function to partition a tensor into windows.

$$\underset{\mathsf{seq,kernel}}{\operatorname{pool}} : \mathbb{R}^{\mathsf{seq}[n]} \to \mathbb{R}^{\mathsf{seq}[n/|\mathsf{kernel}|],\mathsf{kernel}}$$
$$\underset{\mathsf{seq,kernel}}{\operatorname{pool}} X = Y, \text{ where}$$
$$Y_{\mathsf{seq}(i),\mathsf{kernel}(j)} = X_{\mathsf{seq}((i-1)\cdot|\mathsf{kernel}|+j)}.$$

Then we can define aggregations over kernel. We define max-pooling as:

$$\begin{aligned}\text{MaxPool1d}_k\colon \mathbb{R}^{\mathsf{seq}[n]} &\to \mathbb{R}^{\mathsf{seq}[n/k]}\\ \text{MaxPool1d}_k(X) &= \max_{\mathsf{kernel}} \underset{\mathsf{seq,kernel}}{\operatorname{pool}} X\\ |\mathsf{kernel}| &= k\\ \text{MaxPool2d}_{kh,kw}\colon \mathbb{R}^{\mathsf{height}[h]\times\mathsf{width}[w]} &\to \mathbb{R}^{\mathsf{height}[h/kh]\times\mathsf{width}[w/kw]}\\ \text{MaxPool2d}_{kh,kw}(X) &= \max_{\mathsf{kh,kw}} \underset{\mathsf{height,kh}}{\operatorname{pool}} \underset{\mathsf{width,kw}}{\operatorname{pool}} X\\ |\mathsf{kh}| &= kh\\ |\mathsf{kw}| &= kw.\end{aligned}$$

4.6 Normalization layers

Normalization layers are used in all large-scale deep learning models, with different architectures requiring different types of normalization. However, despite their importance, the differences between them are often not clearly communicated. For example, the PyTorch documentation (PyTorch Contributors, 2022) describes all of them using the same equation (where $\epsilon > 0$ is a small constant for numerical stability):

$$Y = \frac{X - \text{mean}(X)}{\sqrt{\text{var}(X) + \epsilon}} \odot \gamma + \beta$$

Wu & He (2018) give essentially the same equation and explain the differences using a combination of equations, words, and pictures. But they do not capture differences in γ and β among different normalization layers.

Critically, the layers do differ by which axes are *standardized* as well as their parameters. We define a single named standardization function as:

$$\underset{\mathsf{ax}}{\text{standardize}} \colon \mathbb{R}^{\mathsf{ax}} \to \mathbb{R}^{\mathsf{ax}}$$

$$\underset{\mathsf{ax}}{\text{standardize}}(X) = \frac{X - \underset{\mathsf{ax}}{\text{mean}}(X)}{\sqrt{\underset{\mathsf{ax}}{\text{var}}(X) + \epsilon}}$$

Then, we can define the three kinds of normalization layers, all with type $\mathbb{R}^{\mathsf{batch} \times \mathsf{chans} \times \mathsf{layer}} \to \mathbb{R}^{\mathsf{batch} \times \mathsf{chans} \times \mathsf{layer}}$. While superficially similar, these functions differ in their standardized axes and their parameter shape.

$$\text{BatchNorm}(X; \gamma, \beta) = \underset{\mathsf{batch},\mathsf{layer}}{\text{standardize}}(X) \odot \gamma + \beta \qquad \gamma, \beta \in \mathbb{R}^{\mathsf{chans}}$$

$$\text{InstanceNorm}(X; \gamma, \beta) = \underset{\mathsf{layer}}{\text{standardize}}(X) \odot \gamma + \beta \qquad \gamma, \beta \in \mathbb{R}^{\mathsf{chans}}$$

$$\text{LayerNorm}(X; \gamma, \beta) = \underset{\mathsf{layer},\mathsf{chans}}{\text{standardize}}(X) \odot \gamma + \beta \qquad \gamma, \beta \in \mathbb{R}^{\mathsf{chans},\mathsf{layer}}$$

5 Differentiation

Let f be a function from order-m tensors to order-n tensors and let $Y = f(X)$. The partial derivatives of Y with respect to X form an order-$(m+n)$ tensor: m "input" axes for the directions in which X could change and n "output" axes for the change in Y.

For example, if f maps from vectors to vectors, then $\frac{\partial Y}{\partial X}$ is a matrix (the Jacobian). But using matrix notation, there are conflicting conventions about whether the first axis is the input axis ("denominator layout") or the output axis ("numerator layout"). The derivative of a function from vectors to matrices or matrices to vectors cannot be represented as a matrix at all, so one must resort to flattening the matrices into vectors.

With non-named index notation, taking derivatives of higher-order tensors with respect to higher-order tensors is not difficult (Laue et al., 2018), but again a convention must be adopted that the input axes come after the output axes, and the output and input axes are separated with a comma.

5.1 Definition

With named tensors, axes are not ordered, so there is no need to establish and remember conventions about whether the input or output axes come first. The only thing we have to worry about is if an input and output axis have the same name, which we can handle using renaming.

Definition 8. Let $f \colon \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{T}}$, where $\mathcal{S}$ and $\mathcal{T}$ are orthogonal, and let $Y = f(X)$. Then the derivative of Y at X is the tensor with shape $\mathcal{S} \times \mathcal{T}$ such that for all $s \in \text{rec}\,\mathcal{S}$ and $t \in \text{rec}\,\mathcal{T}$,

$$\left[\frac{\partial Y}{\partial X}\right]_{s,t} = \frac{\partial Y_t}{\partial X_s}.$$

If X and Y's shapes are not orthogonal, we take the derivative of $Y_{\mathcal{T}\to\mathcal{T}'}$ instead. (It's also possible to rename X, but we think it's easier to think about renaming Y.) Assume $\mathcal{T} = \mathsf{ax}_1 \times \cdots \times \mathsf{ax}_r$. Then for each ax_i, choose a new name ax'_i not in either $\mathcal{S}$ or $\mathcal{T}$, and let $\mathcal{T}' = \mathsf{ax}'_1 \times \cdots \times \mathsf{ax}'_r$. Then we seek the tensor of partial derivatives

$$\left[\frac{\partial Y_{\mathcal{T}\to\mathcal{T}'}}{\partial X}\right]_{s,t'} = \frac{\partial Y_t}{\partial X_s}.$$

5.2 Rules

To compute derivatives, we use the method of differentials (Magnus & Neudecker, 1985). Intuitively, the *differential* of $f(X)$ at X, written $\partial f(X;\Delta X)$, is a tensor with the same shape as $f(X)$ that linearly approximates f at $X+\Delta X$. We omit a formal definition here. Magnus & Neudecker prove that if ∂f exists, it is unique, and

Theorem 1. *If $f\colon \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{T}}$ where $\mathcal{S}$ and $\mathcal{T}$ are orthogonal, and ∂f exists at C, then for all $A \in \mathbb{R}^{\mathcal{S}\times\mathcal{T}}$,*

$$\partial f(C;\Delta X) = A \underset{\mathcal{S}}{\odot} \Delta X \quad (\forall\, \Delta X \in \mathbb{R}^{\mathcal{S}}) \qquad \textit{iff} \qquad \frac{\partial f(X)}{\partial X} = A.$$

Magnus & Neudecker state this result twice, one for vector-to-vector functions and once for matrix-to-matrix functions (but omitting vector-to-matrix and matrix-to-vector functions). Here, we only need to state it once, for functions from tensors to tensors of any shape.

In practice, we can write $\partial f(X;\Delta X)$ simply as ∂f, and ΔX never appears in our calculations. We find differentials using rules like the following:

$$\begin{aligned}
\partial(U+V) &= \partial U + \partial V \\
\partial(U \odot V) &= \partial U \odot V + U \odot \partial V \\
\partial\left(\frac{U}{V}\right) &= \frac{\partial U \odot V - U \odot \partial V}{V^2} \\
\partial \sum_{\mathsf{ax}} U &= \sum_{\mathsf{ax}} \partial U \\
\partial(U \underset{\mathsf{ax}}{\odot} V) &= \partial U \underset{\mathsf{ax}}{\odot} V + U \underset{\mathsf{ax}}{\odot} \partial V \\
\partial U_s &= [\partial U]_s \\
\partial U_{\mathsf{ax}\to\mathsf{ax}'} &= [\partial U]_{\mathsf{ax}\to\mathsf{ax}'}
\end{aligned}$$

We will also make use of the chain rule,

$$\partial f(U) = \frac{\partial f(U)}{\partial U} \underset{\mathcal{U}}{\odot} \partial U \qquad\qquad f\colon \mathbb{R}^{\mathcal{U}} \to \mathbb{R}^{\mathcal{V}}.$$

If we can get the differential of Y into so-called *canonical form*,

$$\partial Y = A \underset{\mathcal{S}}{\odot} \partial X + \text{const.} \tag{1}$$

where "const." stands for terms not depending on ∂X, then by Theorem 1 we have

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

In order to get equations into canonical form, some tricks are useful. First, contractions can be easier to reason about if rewritten as sums of elementwise products:

$$A \underset{\mathsf{ax}}{\odot} B = \sum_{\mathsf{ax}} (A \odot B).$$

Second, renaming can be thought of as contraction with an identity matrix:

$$[I_{\mathsf{ax},\mathsf{ax}'}]_{\mathsf{ax}(i),\mathsf{ax}'(j)} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$
$$A_{\mathsf{ax}\to\mathsf{ax}'} = \sum_{\mathsf{ax}} I_{\mathsf{ax},\mathsf{ax}'} \odot A.$$

5.3 Example

Let us find the differential of the softmax operator:

$$\begin{aligned} Y &= \underset{\mathsf{ax}}{\operatorname{softmax}} X \\ \partial Y &= \partial\left(\frac{\exp X}{\sum\limits_{\mathsf{ax}} \exp X}\right) \\ &= \frac{\exp X \odot \partial X \odot \sum\limits_{\mathsf{ax}} \exp X - \exp X \odot \sum\limits_{\mathsf{ax}} (\exp X \odot \partial X)}{\left(\sum\limits_{\mathsf{ax}} \exp X\right)^2} \\ &= Y \odot (\partial X - Y \underset{\mathsf{ax}}{\odot} \partial X). \end{aligned}$$

Next, use this to find the Jacobian, $\frac{\partial Y}{\partial X}$. Since X and Y have the same shape, we rename Y:

$$\begin{aligned} \partial Y_{\mathsf{ax}\to\mathsf{ax}'} &= [Y \odot (\partial X - \sum_{\mathsf{ax}} Y \odot \partial X)]_{\mathsf{ax}\to\mathsf{ax}'} \\ &= Y_{\mathsf{ax}\to\mathsf{ax}'} \odot (\partial X_{\mathsf{ax}\to\mathsf{ax}'} - \sum_{\mathsf{ax}} Y \odot \partial X) \\ &= Y_{\mathsf{ax}\to\mathsf{ax}'} \odot \left(\sum_{\mathsf{ax}} I_{\mathsf{ax}',\mathsf{ax}} \odot \partial X - \sum_{\mathsf{ax}} Y \odot \partial X\right) \\ &= \sum_{\mathsf{ax}} Y_{\mathsf{ax}\to\mathsf{ax}'} \odot (I_{\mathsf{ax}',\mathsf{ax}} - Y) \odot \partial X \\ \frac{\partial Y_{\mathsf{ax}\to\mathsf{ax}'}}{\partial X} &= Y_{\mathsf{ax}\to\mathsf{ax}'} \odot (I_{\mathsf{ax}',\mathsf{ax}} - Y). \end{aligned}$$

To derive the rule for backpropagation, we assume there is a scalar L that depends on Y and that we already know $\frac{\partial L}{\partial Y}$. We want to find $\frac{\partial L}{\partial X}$.

$$\begin{aligned} \partial L &= \sum_{\mathsf{ax}} \frac{\partial L}{\partial Y} \odot \partial Y \\ &= \sum_{\mathsf{ax}} \frac{\partial L}{\partial Y} \odot Y \odot (\partial X - \sum_{\mathsf{ax}} Y \odot \partial X) \\ &= \sum_{\mathsf{ax}} \frac{\partial L}{\partial Y} \odot Y \odot \partial X - \sum_{\mathsf{ax}} \frac{\partial L}{\partial Y} \odot Y \odot \sum_{\mathsf{ax}} Y \odot \partial X \\ &= \sum_{\mathsf{ax}} \frac{\partial L}{\partial Y} \odot Y \odot \partial X - \sum_{\mathsf{ax}} \left(\sum_{\mathsf{ax}} \frac{\partial L}{\partial Y} \odot Y\right) \odot Y \odot \partial X \\ &= \sum_{\mathsf{ax}} Y \odot \left(\frac{\partial L}{\partial Y} - \sum_{\mathsf{ax}} \frac{\partial L}{\partial Y} \odot Y\right) \odot \partial X \end{aligned}$$

$$\frac{\partial f(Y)}{\partial X} = Y \odot \left(\frac{\partial L}{\partial Y} - \frac{\partial L}{\partial Y} \underset{\mathsf{ax}}{\odot} Y \right).$$

5.4 Lifting

Let $f\colon \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{T}}$, and let f' be its derivative. If $X \in \mathbb{R}^{\mathcal{S} \cup \mathcal{U}}$, where $\mathcal{U}$ is orthogonal to both $\mathcal{S}$ and $\mathcal{T}$, recall that $Y = f(X)$ is defined by:

$$Y_r = f(X_r)$$

Finding the differential of Y is easy:

$$\partial Y_r = f'(X_r) \underset{\mathcal{S}}{\odot} \partial X_r$$
$$\partial Y = f'(X) \underset{\mathcal{S}}{\odot} \partial X.$$

But although f' can be to X using the usual lifting rules, it's not the case that $\frac{\partial Y}{\partial X} = f'(X)$, which would have the wrong shape. The reason is that the contraction is only over $\mathcal{S}$, not $\mathcal{S} \cup \mathcal{U}$. To get this into the form (1):

$$\begin{aligned} \partial Y_{\mathcal{U} \to \mathcal{U}'} &= \sum_{\mathcal{S}} [f'(X) \odot \partial X]_{\mathcal{U} \to \mathcal{U}'} \\ &= \sum_{\mathcal{S}} \sum_{\mathcal{U}} I_{\mathcal{U},\mathcal{U}'} \odot f'(X) \odot \partial X \\ \frac{\partial Y_{\mathcal{U} \to \mathcal{U}'}}{\partial X} &= I_{\mathcal{U},\mathcal{U}'} \odot f'(X). \end{aligned}$$

In general, then, when we lift a function to new axes, we lift its derivative by multiplying by the identity matrix for those axes.

6 Alternatives and Related Work

6.1 Index notations

Among alternatives to standard vector and matrix notation, the most common one, which scales much better to more than two axes, is index notation as used in physics (Ricci & Levi-Civita, 1901), including the Einstein summation convention. Related notations are used in other fields as well (Harshman, 2001). In this notation, axes are ordered, and every equation is written in terms of tensor components. If an index appears on both sides of an equation, then the equation must hold for each value of the index, and if an index appears twice on one side and not on the other, there is an implicit summation over that index.

$$\text{Attention}\colon \mathbb{R}^{d_k} \times \mathbb{R}^{n \times d_k} \times \mathbb{R}^{n \times d_v} \to \mathbb{R}^{d_v}$$
$$[\text{Attention}(Q, K, V)]_k = \underset{i}{\text{softmax}} \left(\frac{Q_j K_{ij}}{\sqrt{d_k}} \right) V_{ik}.$$

Because k appears on both sides, the equation must hold over all values of this index. But because i and j occur twice on only the right-hand side, they are both summed over. We would have to define exactly what the i under the softmax means (i is bound inside the softmax and free outside it), and since softmax doesn't distribute over addition, we would need to modify the summation convention so that the summation over j occurs inside the softmax.

Aside from these correctable issues, this notation is concise and unambiguous. But it does not solve the main problem we set out to solve, which is that ordered axes force the author and reader to remember the purpose of each axis. The indices do act as symbolic names for axes (indeed, in *abstract* index notation (Penrose & Rindler, 1984), they really are symbols, not variables), but they are temporary names; they could be totally different in the next equation. It would be up to the author to choose to use consistent names, and to do so correctly.

A second issue is that because it depends on repetition of indices to work, index notation can be more verbose than our notation, particularly for reductions and contractions:

$$C = \max_i A_i \qquad C = \max_{\mathsf{ax}} A$$

$$C = A_i B_i \qquad C = A \underset{\mathsf{ax}}{\odot} B.$$

Finally, index notation requires us to write out all indices explicitly. So if we wanted to lift attention to minibatches ($b \in [B]$), multiple heads ($h \in [H]$) and multiple query tokens ($i' \in [n']$), we would write:

$$\text{Attention} \colon \mathbb{R}^{B \times H \times n' \times d_k} \times \mathbb{R}^{B \times H \times n \times d_k} \times \mathbb{R}^{B \times H \times n \times d_v} \to \mathbb{R}^{B \times H \times n' \times d_v}$$

$$[\text{Attention}(Q, K, V)]_{bhi'k} = \operatorname*{softmax}_i \left(\frac{Q_{bhi'j} K_{bhij}}{\sqrt{d_k}} \right) V_{bhik}.$$

We could adopt a convention that lifts a function on tensors to tensors that have extra axes to the *left*, but such conventions tend to lead to messy reordering and squeezing/unsqueezing of axes. Named axes make such conventions unnecessary.

6.2 Graphical notations

In the graphical notation of Penrose (1971), a node in the graph stands for a tensor, and its incident edges stand for its indices. The edges are ordered from left to right. An edge connecting two nodes denotes contraction. The notation of Alsberg (1997) is similar, except that edges are named, not ordered.

Graphs are commonly used in machine learning for representing probability models (Koller & Friedman, 2009). A node in the graph stands for a random variable, and an edge or hyperedge stands for a dependency between variables. If random variables have finite domains, then a (hyper)edge with r endpoints can be thought of as an r-th order tensor. A graph can then be thought of as a product and contraction. Extensions that allow for a choice between two subgraphs (e.g., Minka & Winn, 2008) can be thought of as addition.

Our assessment of graphical notations like these is that, on the positive side, they have obvious value for visualization, and they at least have the potential to represent indices in a purely unordered way. On the negative side, these notations seem best suited for representing linear functions, and even for this purpose, some other practical considerations are that drawing pictures requires more effort from the author, and that pictures will have a less transparent relationship with their implementation in most programming languages.

6.3 Relational algebra

In relational algebra (Codd, 1970), the basic objects are sets of r-tuples, which could be thought of as tensors of order r with Boolean-valued entries. In the original formulation, the members of the tuples, which correspond to axes, were both named *and* ordered, although later definitions (e.g. Pirotte, 1982) made them unordered.

Probabilistic variants of relational algebra also exist (e.g. Dey & Sarkar, 1996; Fuhr & Rölleke, 1997), whose relations would correspond to tensors of probabilities.

While relational algebra and tensor notations are designed for totally different purposes, the notation of relational algebra generally has a similar flavor to ours (for example, our contraction operator is similar to the $\bowtie$ operator, and our renaming operator is the same as the ρ operator).

7 Conclusions

Named tensor notation is a system of formal notation for representing operations between tensors in a non-ambiguous way while remaining intuitive for practitioners. The system is motivated by challenges that arise from taking notation designed for applied linear algebra and using it for representing neural networks, as demonstrated through examples of canonical deep-learning components such as attention and layer normalization. However, named tensors are not limited to specifying neural networks. We have also explained how to integrate our notation with Magnus & Neudecker (1985)'s method of differentials for matrix

calculus. While there are other conventions that such as index notation that have some usage in the machine learning community, these conventions either lack the conciseness of named tensors or are not well-suited to non-linear operations. For these reasons, we encourage members of the machine learning community to try out named tensor notation for teaching, research, and software documentation.

References

Bjørn K. Alsberg. A diagram notation for n-mode array equations. *Journal of Chemometrics*, 11:251–266, 1997.

Matt Bauman. Axisarrays, 2018. URL `https://github.com/JuliaArrays/AxisArrays.jl`. Open-source software.

Tongfei Chen. Typesafe abstractions for tensor operations. In *Proceedings of the 8th ACM SIGPLAN International Symposium on Scala*, SCALA 2017, pp. 45–50, 2017. doi: 10.1145/3136000.3136001. URL `http://doi.acm.org/10.1145/3136000.3136001`.

E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, jun 1970. doi: 10.1145/362384.362685.

Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Trans. Database Syst.*, 21(3):339–369, September 1996. doi: 10.1145/232753.232796.

Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, January 1997. doi: 10.1145/239041.239045.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Richard A. Harshman. An index formalism that generalizes the capabilities of matrix notation and algebra to n-way arrays. *Journal of Chemometrics*, 15:689–714, 2001. doi: 10.1002/cem.665.

Stephan Hoyer and Joe Hamman. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1):10, 2017. doi: http://doi.org/10.5334/jors.148.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

Soeren Laue, Matthias Mitterreiter, and Joachim Giesen. Computing higher order derivatives of matrix and tensor expressions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31, pp. 2750–2759. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/0a1bf96b7165e962e90cb14648c9462d-Paper.pdf`.

Dougal Maclaurin, Alexey Radul, Matthew J. Johnson, and Dimitrios Vytiniotis. Dex: array programming with typed indices. In *NeurIPS Workshop on Program Transformations for ML*, 2019. URL `https://openreview.net/forum?id=rJxd7vsWPS`.

Jan R. Magnus and H. Neudecker. Matrix differential calculus with applications to simple, Hadamard, and Kronecker products. *Journal of Mathematical Psychology*, 29(4):474–492, 1985. doi: https://doi.org/10.1016/0022-2496(85)90006-9. URL `http://www.sciencedirect.com/science/article/pii/0022249685900069`.

Tom Minka and John Winn. Gates. In *Proc. NeurIPS*, pp. 1073–1080, 2008. URL `https://papers.nips.cc/paper/3379-gates`.

R. Penrose and W. Rindler. *Spinors and space-time*, volume 1. Cambridge University Press, 1984.

Roger Penrose. Applications of negative dimensional tensors. In D. J. A. Welsh (ed.), *Combinatorial Mathematics and its Applications*, pp. 221–244. Academic Press, 1971.

Alain Pirotte. A precise definition of basic relational notations and of the relational algebra. *ACM SIGMOD Record*, 13(1):30–45, 1982. doi: 10.1145/984514.984516.

PyTorch Contributors. PyTorch documentation, 2022. URL `https://pytorch.org/docs/1.12/`. version 1.12.

G. Ricci and T. Levi-Civita. Méthodes de calcul différentiel absolu et leurs applications. *Mathematische Annalen*, 54:125–201, 1901.

Alexander Rush. Named tensors, 2019. URL `https://github.com/harvardnlp/NamedTensor`. Open-source software.

Nishant Sinha. Tensor shape (annotation) library, 2018. URL `https://github.com/ofnote/tsalib`. Open-source software.

Torch Contributors. Named tensors, 2019. URL `https://pytorch.org/docs/stable/named_tensor.html`. PyTorch documentation.

L. R. Tucker. The extension of factor analysis to three-dimensional matrices. In H. Gulliksen and N. Frederiksen (eds.), *Contributions to Mathematical Psychology*, pp. 110–127. Holt, Rinehart and Winston, New York, 1964.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Yuxin Wu and Kaiming He. Group normalization. In *Proc. ECCV*, 2018.

A Extended Examples

A.1 Transformer

We define a Transformer used autoregressively as a language model. The input is a sequence of one-hot vectors, from which we compute word embeddings and positional encodings:

$$I \in \{0,1\}^{\mathsf{seq}\times\mathsf{vocab}} \qquad \sum_{\mathsf{vocab}} I = 1$$

$$W = (E \underset{\mathsf{vocab}}{\odot} I)\sqrt{|\mathsf{layer}|} \qquad E \in \mathbb{R}^{\mathsf{vocab}\times\mathsf{layer}}$$

$$P \in \mathbb{R}^{\mathsf{seq}\times\mathsf{layer}}$$

$$P_{\mathsf{seq}(p),\mathsf{layer}(i)} = \begin{cases} \sin((p-1)/10000^{(i-1)/|\mathsf{layer}|}) & i \text{ odd} \\ \cos((p-1)/10000^{(i-2)/|\mathsf{layer}|}) & i \text{ even.} \end{cases}$$

Then we use L layers of self-attention and feed-forward neural networks:

$$\begin{aligned} X^0 &= W + P \\ T^1 &= \mathrm{LayerNorm}^1(\mathrm{SelfAtt}^1(X^0)) + X^0 \\ X^1 &= \mathrm{LayerNorm}^{1'}(\mathrm{FFN}^1(T^1)) + T^1 \\ &\vdots \\ T^L &= \mathrm{LayerNorm}^L(\mathrm{SelfAtt}^L(X^{L-1})) + X^{L-1} \\ X^L &= \mathrm{LayerNorm}^{L'}(\mathrm{FFN}^L(T^L)) + T^L \\ O &= \underset{\mathsf{vocab}}{\mathrm{softmax}}(E \underset{\mathsf{layer}}{\odot} X^L) \end{aligned}$$

where LayerNorm, SelfAtt and FFN are defined below.

Layer normalization ($l = 1, 1', \ldots, L, L'$):

$$\begin{aligned}\text{LayerNorm}^l &: \mathbb{R}^{\mathsf{layer}} \to \mathbb{R}^{\mathsf{layer}} \\ \text{LayerNorm}^l(X) &= \underset{\mathsf{layer}}{\text{XNorm}}(X; \beta^l, \gamma^l).\end{aligned}$$

We defined attention in §4.3; the Transformer uses multi-head self-attention, in which queries, keys, and values are all computed from the same sequence.

$$\begin{aligned}\text{SelfAtt}^l &: \mathbb{R}^{\mathsf{seq}\times\mathsf{layer}} \to \mathbb{R}^{\mathsf{seq}\times\mathsf{layer}} \\ \text{SelfAtt}^l(X) &= Y\end{aligned}$$

where

$$\begin{aligned}|\mathsf{seq}| &= |\mathsf{seq}'| \\ |\mathsf{key}| = |\mathsf{val}| &= |\mathsf{layer}|/|\mathsf{heads}| \\ Q &= W^{l,Q} \underset{\mathsf{layer}}{\odot} X_{\mathsf{seq}\to\mathsf{seq}'} && W^{l,Q} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{layer}\times\mathsf{key}} \\ K &= W^{l,K} \underset{\mathsf{layer}}{\odot} X && W^{l,K} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{layer}\times\mathsf{key}} \\ V &= W^{l,V} \underset{\mathsf{layer}}{\odot} X && W^{l,V} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{layer}\times\mathsf{val}} \\ M &\in \mathbb{R}^{\mathsf{seq}\times\mathsf{seq}'} \\ M_{\mathsf{seq}(i),\mathsf{seq}'(j)} &= \begin{cases} 0 & i \le j \\ -\infty & \text{otherwise} \end{cases} \\ Y &= W^{l,O} \underset{\substack{\mathsf{heads}\\ \mathsf{val}}}{\odot} \text{Attention}(Q, K, V, M)_{\mathsf{seq}'\to\mathsf{seq}} && W^{l,O} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{val}\times\mathsf{layer}}\end{aligned}$$

Feedforward neural networks:

$$\begin{aligned}\text{FFN}^l &: \mathbb{R}^{\mathsf{layer}} \to \mathbb{R}^{\mathsf{layer}} \\ \text{FFN}^l(X) &= X^2\end{aligned}$$

where

$$\begin{aligned}X^1 &= \text{relu}(W^{l,1} \underset{\mathsf{layer}}{\odot} X + b^{l,1}) && W^{l,1} \in \mathbb{R}^{\mathsf{hidden}\times\mathsf{layer}} && b^{l,1} \in \mathbb{R}^{\mathsf{hidden}} \\ X^2 &= \text{relu}(W^{l,2} \underset{\mathsf{hidden}}{\odot} X^1 + b^{l,2}) && W^{l,2} \in \mathbb{R}^{\mathsf{layer}\times\mathsf{hidden}} && b^{l,2} \in \mathbb{R}^{\mathsf{hidden}}.\end{aligned}$$

A.2 LeNet

$$\begin{aligned}X^0 &\in \mathbb{R}^{\mathsf{batch}\times\mathsf{chans}[c_0]\times\mathsf{height}\times\mathsf{width}} \\ T^1 &= \text{relu}(\text{Conv}^1(X^0)) \\ X^1 &= \text{MaxPool}^1(T^1) \\ T^2 &= \text{relu}(\text{Conv}^2(X^1)) \\ X^2 &= \text{MaxPool}^2(T^2)_{(\mathsf{height},\mathsf{width},\mathsf{chans})\to\mathsf{layer}} \\ X^3 &= \text{relu}(W^3 \underset{\mathsf{layer}}{\odot} X^2 + b^3) && W^3 \in \mathbb{R}^{\mathsf{hidden}\times\mathsf{layer}} && b^3 \in \mathbb{R}^{\mathsf{hidden}}\end{aligned}$$

$$O = \underset{\mathsf{classes}}{\mathrm{softmax}}(W^4 \underset{\mathsf{hidden}}{\odot} X^3 + b^4) \qquad W^4 \in \mathbb{R}^{\mathsf{classes} \times \mathsf{hidden}} \qquad b^4 \in \mathbb{R}^{\mathsf{classes}}$$

As an alternative to the flattening operation in the equation for X^2, we could have written

$$\begin{aligned} X^2 &= \mathrm{MaxPool}^2(T^2) \\ X^3 &= \mathrm{relu}(W^3 \underset{\substack{\mathsf{height} \\ \mathsf{width} \\ \mathsf{chans}}}{\odot} X^2 + b^3) \qquad W^3 \in \mathbb{R}^{\mathsf{hidden} \times \mathsf{height} \times \mathsf{width} \times \mathsf{chans}}. \end{aligned}$$

The convolution and pooling operations are defined as follows:

$$\mathrm{Conv}^l(X) = \mathrm{Conv2d}(X; W^l, b^l)_{\mathsf{chans}' \to \mathsf{chans}}$$

where

$$\begin{aligned} W^l &\in \mathbb{R}^{\mathsf{chans}'[c_l] \times \mathsf{chans}[c_{l-1}] \times \mathsf{kh}[kh_l] \times \mathsf{kw}[kw_l]} \\ b^l &\in \mathbb{R}^{\mathsf{chans}'[c_l]} \end{aligned}$$

and

$$\mathrm{MaxPool}^l(X) = \mathrm{MaxPool2d}_{ph^l, ph^l}(X).$$