Named Tensor Notation

David Chiang, Sasha Rush, and Boaz Barak

December 18, 2020

Abstract

We propose a notation for tensors with named axes. For example, instead of writing $A \in \mathbb{R}^{b \times w \times h \times c}$ for an order-4 tensor corresponding to a batch of images, we use $A \in \mathbb{R}^{\mathsf{batch} \times \mathsf{width} \times \mathsf{height} \times \mathsf{channel}}$. Such a tensor can be indexed using these names, as in $A_{\mathsf{batch}(25),\mathsf{width}(12),\mathsf{height}(30),\mathsf{channel}(1)}$, which refers to the same index as $A_{\mathsf{channel}(1),\mathsf{width}(12),\mathsf{height}(30),\mathsf{batch}(25)}$. Named notation relieves the author, reader, and future implementers from needing to keep track of the order of parameters. It also makes it easier to express "slices" of tensors, such as $A_{\mathsf{batch}(17)}$ or $A_{\mathsf{channel}(\text{`Red'})}$, and to seamlessly extend operations on low-order tensors to higher order ones (e.g., extend an operation on images to batches on images, or extend the attention mechanism to multiple tokens and heads).

In this document we present the notation, and illustrate it through several examples from modern machine learning, including both building blocks such as Attention and Convolutions, and full network architectures such as Transformers and LeNet. Most of our proposals are not original, and we borrow ideas from several works and packages. We hope that by putting together this document, we can encourage more authors to use named tensors, resulting in clearer papers and less bug-prone implementations. The source code for this document can be found at `https://github.com/namedtensor/notation/`. We invite anyone to make comments on this proposal by submitting issues or pull requests on this repository.

1 Introduction

Most papers about neural networks use the notation of vectors and matrices from applied linear algebra. This notation is very well-suited to talking about vector spaces, but becomes cumbersome and confusing when extended to the high-order tensors of diverse axes used in modern neural networks. Consider the following equation for the *Attention* mechanism (**?**):

$$\text{Att}(Q\ ,\ K\ ,\ V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right) V\ . \tag{1}$$

where Q, K, and V are sequences of query, key, and value vectors packed into matrices. Does the product $QK^\top$ sum over the sequence, or over the query/key features? We would need to know the sizes of Q, K, and V to know that it's taken over the query/key features. Is the softmax taken over the query sequence or the key sequence? The usual notation doesn't even offer a way to answer this question. With multiple attention heads and multiple sentences in a minibatch, Q and K become tensors with four separate axes/dimensions, and most papers skip a full description with all these details.

In this document we propose an intuitive mathematical notation for Tensors with *named axes*. The notation has a formal underpinning, but it is intuitive enough that researchers in machine learning should be able to follow it without needing to look any concepts up. As a preview, in our notation, Equation (1) becomes

$$\mathrm{Att} : \mathbb{R}^{\mathsf{key}} {\times} \mathbb{R}^{\mathsf{seq} \times \mathsf{key}} {\times} \mathbb{R}^{\mathsf{seq}} \to \mathbb{R}$$

$$\mathrm{Att}(\, Q \,,\, K \,,\, V\,) \quad = \underset{\mathsf{seq}}{\mathrm{softmax}} \left(\frac{Q \underset{\mathsf{key}}{\cdot} K}{\sqrt{|\mathsf{key}|}} \right) \underset{\mathsf{seq}}{\cdot} V \,. \tag{2}$$

Unlike (1), in Equation (2) the reader does not need to remember whether it is the columns or the rows of K that correspond to distinct queries, since the notation makes it clear that the dot product inside the soft max is taken along the key axis, resulting in a vector in $\mathbb{R}^{\mathsf{seq}}$. The softmax operation is then applied to every element of this vector, and we take a dot product of it with V along the seqaxis. The formula generalizes *as written* to the case that the we have multiple heads, tokens, and a minibatch. This will correspond to adding a seq axis to Q (to allow for a sequence of input tokens), and adding the head and batch axes to all input tensors. Our convention is that these additional input axes "pass through" to the output. Thus instead of a single number, the output will be a tensor in $\mathbb{R}^{\mathsf{seq} \times \mathsf{head} \times \mathsf{batch}}$, each coordinate of which corresponds to invoking the original operation on the corresponding "slice" of the input tensors. The names ensure that we can properly align these slices.

Our notation is inspired by several libraries that have been developed to identify axes by *name* instead: Nexus (**?**), tsalib (**?**), NamedTensor (**?**), named tensors in PyTorch (**?**), and Dex (**?**). However, in this work our focus is on mathematical notation. We hope that widespread adoption of named tensors in papers will lead to their adoption in code as well. We welcome comments on this document, as well as additional examples, via issues or pull requests on the repository `https://github.com/namedtensor/notation/`.

2 Summary of notation

We now provide a very quick overview of the proposed notation, leaving detailed examples to §?? and formal mathematical definitions to §??. A standard ("unnamed") vector, matrix, or tensors is simply a map from one or more intervals of integers to the real numbers. For example, a length n vector is a map $v \in \mathbb{R}^{[n]}$, where $[n] = \{1, 2, \ldots, n\}$, an $n \times m$ matrix is a map $A \in \mathbb{R}^{[n] \times [m]}$, an $n \times m \times k$ three-tensor is a map of the form $T \in \mathbb{R}^{[n] \times [m] \times [k]}$ and so on and so forth. For a tensor $T \in \mathbb{R}^{[n] \times [m] \times [k]}$ and $(i, j, k) \in [n] \times [m] \times [k]$, we typically denote by $T_{i,j,k}$ the real number that (i, j, k) is mapped to by T.

Named tensors are maps from *axes* to the real number. For example, an image might be represented as a named tensor $I \in \mathbb{R}^{\text{width} \times \text{height} \times \text{channel}}$. You can think of the axes as intervals of integers,[1] but one endowed with a *type*, in the sense that (for example) the 7th element of the width axis, which we denote by width(7), is considered distinct from the 7th element of the height axis. In cases where the sizes of an ax is not fixed or can be inferred from context, we use names such as width, height, channel. Otherwise, we use names such as width[w], height[h], channel[c] (with $w, h, c \in \mathbb{N}$) to specify a particular sizes, and so write $I \in \mathbb{R}^{\text{width}[w] \times \text{height}[h] \times \text{channel}[c]}$. If x, y, a is a point in (width, height, channel) coordinates, then we use $I_{\text{width}[x],\text{height}[y],\text{channel}[c]}$ to denote the corresponding value of I. This notation is unordered, and so $I_{\text{channel}[c],\text{height}[y],\text{width}[x]}$ denotes the same value. Similarly, $\mathbb{R}^{\text{width} \times \text{height} \times \text{channel}}$ is the same set as $\mathbb{R}^{\text{channel} \times \text{height} \times \text{width}}$. We also use partial indexing to denote *slices*, and so, for example, $I_{\text{width}[22]}$ denotes the $x = 22$ slice of the image I.

Operations on tensors. An *operator* on tensors is a function F mapping one or more tensors into a tensor. The *signature* of F corresponds to the types (names of axes) of each of the input tensors and of the output tensors. If some of the input tensors have additional axes that are not contained in the signature, then these would be added to the output tensors in the following way. If any of those additional axes are in only one of the input tensors, then they are considered *dangling axes* and passed as they are to the output tensors. For example, if $F : \mathbb{R}^{\text{width} \times \text{height} \times \text{channel}} \to \mathbb{R}^{\text{layer}}$ is an operator, and we pass to F a tensor $X \in \mathbb{R}^{\text{width} \times \text{height} \times \text{channel} \times \text{batch}}$, then batch is a dangling axis, and $F(X)$ is the tensor Y in $R^{\text{layer} \times \text{batch}}$ such that for every batch index i of X (for which we sometimes use the notation $i \in \text{batch}(X)$), $Y_{\text{batch}(i)} = F(X_{\text{batch}(i)})$. If two or more of the input tensor share the same axis (i.e., with the same name), then these are considered *aligned* axes. This axis ax should have the same support in all the input tensors, and the output will contain ax with the slice ax(i) of the output corresponding to the invocation of F on the slices ax(i) of all inputs with the axis ax. For example, if F is the operation that takes two tensors $A, B \in \mathbb{R}^{\text{width} \times \text{height}}$ and produces some number $F(A, B)$ corresponding to their

---

[1]Though we will sometimes include axes ranging over sets such as {'Red', 'Green', 'Blue'} for RGB channels or words, or an axis corresponding to all words in some dictionary or collection.

similarity, then if $X, W \in \mathbb{R}^{\mathsf{width} \times \mathsf{height} \times \mathsf{channel} \times \mathsf{batch}}$ having the same support over the channel and batch axes, then $F(X, W)$ will be a tensor $Y \in \mathbb{R}^{\mathsf{channel} \times \mathsf{batch}}$ such that for every c in the support of channel and b in the support of batch, $F(X, W)_{\mathsf{channel}(c), \mathsf{batch}(b)} = F(X_{\mathsf{channel}(c), \mathsf{batch}(b)}, W_{\mathsf{channel}(c), \mathsf{batch}(b)})$.

Contractions. Named tensors allow us to explicitly state which axis *contractions* such as sum, dot product, mean, variance, etc. operate on. For example, if $X \in \mathbb{R}^{\mathsf{width} \times \mathsf{height} \times \mathsf{channel} \times \mathsf{batch}}$ then $\sum_{\mathsf{batch}} X = \sum_{b \in \mathsf{batch}(X)} X_{\mathsf{batch}(b)}$ is the tensor in $\mathbb{R}^{\mathsf{width} \times \mathsf{height} \times \mathsf{channel}}$ which is sum of all images across all batches, while $\sum_{\mathsf{channel}} X = \sum_{c \in \mathsf{channel}(C)} X_{\mathsf{channel}(c)}$ is the tensor in $\mathbb{R}^{\mathsf{width} \times \mathsf{height} \times \mathsf{batch}}$ which denotes the sum of the channels for every pixel and batch index. We also use similar notation for the dot product. For example, if $X \in \mathbb{R}^{\mathsf{layer} \times \mathsf{batch}}$ and $W \in \mathbb{R}^{\mathsf{layer} \times \mathsf{hidden}}$ then $X \underset{\mathsf{layer}}{\cdot} W$ is the tensor Y in $\mathbb{R}^{\mathsf{hidden} \times \mathsf{batch}}$ such that for every $b \in \mathsf{batch}(X)$ and $h \in \mathsf{hidden}(W)$, $Y_{\mathsf{batch}(b), \mathsf{hidden}(h)} = \sum_{\ell \in \mathsf{layer}} X_{\mathsf{layer}(\ell)} W_{\mathsf{layer}(\ell)}$. In other words, $X \underset{\mathsf{layer}}{\cdot} W$ equals to the matrix product WX where we think of W as a $\mathsf{hidden} \times \mathsf{layer}$ matrix and of X as a $\mathsf{layer} \times \mathsf{batch}$ matrix.

3   Overview of our notation

An *axis* with name ax, which we denote by ax, is a set of pairs $\mathsf{ax} = \{(\mathrm{ax}, i) | i \in X\}$ where X is some ordered set (most typically, and by default, X is an interval of integers starting with 1). The set X is known as the *index set* of ax. We use the notation $\mathsf{ax}(i)$ for the pair (ax, i) and $\mathsf{ax}[n]$ to denote the set $\{(\mathrm{ax}, i) | i \in [n]\}$. More generally for every subset S of ax's index set, we denote the set $\{(\mathrm{ax}, i) | i \in S\}$ by $\mathsf{ax}[S]$.

If foo and bar are axes, a (real-valued) *named tensor* over the axes foo, bar is a function A that maps pairs $(i, j) \in \mathsf{foo}[S] \times \mathsf{foo}[T]$ into a real number, where S, T are some subsets of foo and bar's index sets respectively. We write this as $A \in \mathbb{R}^{\mathsf{foo} \times \mathsf{bar}}$. We use the notation $A \in \mathbb{R}^{\mathsf{foo}[f], \mathsf{bar}[b]}$ when we want to explicitly state the lengths of each of A's axes, and indicate that A is defined over all $(i, j) \in \mathsf{foo}[f] \times \mathsf{bar}[b]$. For example, here is a tensor with shape $\mathsf{foo}[2] \times \mathsf{bar}[3]$.

$$A = \mathsf{foo} \begin{matrix} \mathsf{bar} \\ \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \end{bmatrix} \end{matrix}.$$

We access elements of A using subscripts: $A_{\mathsf{foo}(1), \mathsf{bar}(3)} = 4$. We also allow partial indexing:

$$A_{\mathsf{foo}(1)} = \begin{matrix} \mathsf{bar} \\ \begin{bmatrix} 3 & 1 & 4 \end{bmatrix} \end{matrix} \qquad\qquad A_{\mathsf{bar}(3)} = \begin{matrix} \mathsf{foo} \\ \begin{bmatrix} 4 & 9 \end{bmatrix} \end{matrix}.$$

We use uppercase italic letters for variables standing for named tensors. We don't mind if you use another convention, but urge you not to use different

styles for tensors and their elements. For example, if $\mathbf{A}$ is a tensor, then an element of $\mathbf{A}$ is written as $\mathbf{A}_{\mathsf{foo}(2),\mathsf{bar}(3)}$ – not $A_{\mathsf{foo}(2),\mathsf{bar}(3)}$ or $a_{\mathsf{foo}(2),\mathsf{bar}(3)}$.

Just as the set of all size-n real vectors is written $\mathbb{R}^n$, and the set of all $m \times n$ real matrices is often written $\mathbb{R}^{m \times n}$, we write $\mathbb{R}^{\mathsf{foo}[2] \times \mathsf{bar}[3]}$ for the set of all tensors with shape $\mathsf{foo}[2] \times \mathsf{bar}[3]$.

In many contexts, an axis name is used with only one size. In this case, you can simply write foo for the unique axis with name foo, as in $\mathbb{R}^{\mathsf{foo} \times \mathsf{bar}}$. It's common to leave the size of an axis unspecified at first, and specify its size later (like in a section or appendix on experimental details). To do this, write $|\mathsf{foo}| = 2$, $|\mathsf{bar}| = 3$.

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, a minibatch of sentences, each of which is a sequence of one-hot vectors, would be represented by a tensor with three axes, which we might name batch, seq, and vocab. Please see §**??** for more examples.

3.1 Named tensor operations

3.1.1 Elementwise operations

Any function from scalars to scalars can be applied elementwise to a named tensor:

$$\exp A = \mathsf{foo}\begin{array}{c}\mathsf{bar}\\\begin{bmatrix}\exp 3 & \exp 1 & \exp 4\\\exp 1 & \exp 5 & \exp 9\end{bmatrix}\end{array}.$$

More elementwise unary operations:

$$\begin{array}{ll}
kA & \text{scalar multiplication by } k\\
-A & \text{negation}\\
A^k & \text{elementwise exponentiation}\\
\sqrt{A} & \text{elementwise square root}\\
\exp A & \text{elementwise exponential function}\\
\tanh A & \text{hyperbolic tangent}\\
\sigma(A) & \text{logistic sigmoid}\\
\mathrm{relu}(A) & \text{rectified linear unit}
\end{array}$$

Any function or operator that takes two scalar arguments can be applied elementwise to two named tensors with the same shape. If A is as above and

$$B = \mathsf{foo}\begin{array}{c}\mathsf{bar}\\\begin{bmatrix}2 & 7 & 1\\8 & 2 & 8\end{bmatrix}\end{array}$$

then

$$A + B = \mathsf{foo}\begin{array}{c}\mathsf{bar}\\\begin{bmatrix}3+2 & 1+7 & 4+1\\1+8 & 5+2 & 9+8\end{bmatrix}\end{array}.$$

But things get more complicated when A and B don't have the same shape. If A and B each have an axis with the same name (and size), the two axes are *aligned*, as above. But if A has an axis named foo and B doesn't, then we do *broadcasting*, which means effectively that we replace B with a new tensor B' that contains a copy of B for every value of axis foo.

$$A + 1 = \mathsf{foo}\overset{\mathsf{bar}}{\begin{bmatrix} 3+1 & 1+1 & 4+1 \\ 1+1 & 5+1 & 9+1 \end{bmatrix}}$$

$$A + B_{\mathsf{foo}(1)} = \mathsf{foo}\overset{\mathsf{bar}}{\begin{bmatrix} 3+2 & 1+7 & 4+1 \\ 1+2 & 5+7 & 9+1 \end{bmatrix}}$$

$$A + B_{\mathsf{bar}(3)} = \mathsf{foo}\overset{\mathsf{bar}}{\begin{bmatrix} 3+1 & 1+1 & 4+1 \\ 1+8 & 5+8 & 9+8 \end{bmatrix}}.$$

Similarly, if B has an axis named foo and A doesn't, then we effectively replace A with a new tensor A' that contains a copy of A for every value of axis foo. If you've programmed with NumPy or any of its derivatives, this should be unsurprising to you.

More elementwise binary operations:

$$
\begin{array}{ll}
A + B & \text{addition} \\
A - B & \text{subtraction} \\
A \odot B & \text{elementwise (Hadamard) product} \\
\dfrac{A}{B} & \text{elementwise division} \\
\max\{A, B\} & \text{elementwise maximum} \\
\min\{A, B\} & \text{elementwise minimum}
\end{array}
$$

3.1.2 Reductions

The same rules for alignment and broadcasting apply to functions that take tensor as arguments or return tensors. The gory details are in §**??**, but we present the most important subcases here. The first is *reductions*, which are functions from vectors to scalars. Unlike with functions on scalars, we always have to specify which axis these functions apply to, using a subscript. (This is equivalent to the `axis` argument in NumPy and `dim` in PyTorch.)

For example, using the same example tensor A from above,

$$\sum_{\mathsf{foo}} A = \overset{\mathsf{bar}}{\begin{bmatrix} 3+1 & 1+5 & 4+9 \end{bmatrix}}$$

$$\sum_{\mathsf{bar}} A = \overset{\mathsf{foo}}{\begin{bmatrix} 3+1+4 & 1+5+9 \end{bmatrix}}.$$

More reductions: If A has an axis $\mathsf{foo}[I]$, then

$$\sum_{\mathsf{foo}} A = \sum_{i \in I} A_{\mathsf{foo}(i)} = \overset{\mathsf{bar}}{\begin{bmatrix} 4 & 6 & 13 \end{bmatrix}}$$

$$\operatorname*{norm}_{\mathsf{foo}} A = \sqrt{\sum_{\mathsf{foo}} A^2} = \overset{\mathsf{bar}}{\begin{bmatrix} \sqrt{10} & \sqrt{26} & \sqrt{97} \end{bmatrix}}$$

$$\min_{\mathsf{foo}} A = \min_{i \in I} A_{\mathsf{foo}(i)} = \overset{\mathsf{bar}}{\begin{bmatrix} 1 & 1 & 4 \end{bmatrix}}$$

$$\max_{\mathsf{foo}} A = \max_{i \in I} A_{\mathsf{foo}(i)} = \overset{\mathsf{bar}}{\begin{bmatrix} 3 & 5 & 9 \end{bmatrix}}$$

$$\operatorname*{mean}_{\mathsf{foo}} A = \frac{1}{|I|} A = \overset{\mathsf{bar}}{\begin{bmatrix} 2 & 3 & 6.5 \end{bmatrix}}$$

$$\operatorname*{var}_{\mathsf{foo}} A = \frac{1}{|I|} \sum_{\mathsf{ax}} (A - \operatorname*{mean}_{\mathsf{foo}} A)^2 = \overset{\mathsf{bar}}{\begin{bmatrix} 1 & 4 & 6.25 \end{bmatrix}}$$

(Note that max and min are overloaded; with multiple arguments and no subscript, they are elementwise, and with a single argument and a subscript, they are reductions.)

You can also write multiple names to perform the reduction over multiple axes at once.

3.1.3   Contraction

The vector dot product (inner product) is a function from *two* vectors to a scalar, which generalizes to named tensors to give the ubiquitous *contraction* operator, which performs elementwise multiplication, then sums along an axis. It can be used, for example, for matrix multiplication:

$$C = \mathsf{bar} \overset{\mathsf{baz}}{\begin{bmatrix} 1 & -1 \\ 2 & -2 \\ 3 & -3 \end{bmatrix}}$$

$$A \underset{\mathsf{bar}}{\cdot} C = \mathsf{foo} \overset{\mathsf{baz}}{\begin{bmatrix} 17 & -17 \\ 53 & -53 \end{bmatrix}}$$

However, note that (like vector dot-product, but unlike matrix multiplication) this operator is commutative, but not associative! Specifically, if

$$A \in \mathbb{R}^{\mathsf{foo}[m]}$$

$$B \in \mathbb{R}^{\mathsf{foo}[m] \times \mathsf{bar}[n]}$$

$$C \in \mathbb{R}^{\mathsf{foo}[m] \times \mathsf{bar}[n]}$$

then $(A \underset{\mathsf{foo}}{\cdot} B) \underset{\mathsf{bar}}{\cdot} C$ and $A \underset{\mathsf{foo}}{\cdot} (B \underset{\mathsf{bar}}{\cdot} C)$ don't even have the same shape.

3.1.4 Vectors to vectors

A very common example of a function from vectors to vectors is the softmax:

$$\underset{\mathsf{foo}}{\mathrm{softmax}}\, A = \frac{\exp A}{\sum_{\mathsf{foo}} \exp A} \approx \mathsf{foo}\begin{array}{c}\mathsf{bar}\\\begin{bmatrix}0.881 & 0.018 & 0.007\\0.119 & 0.982 & 0.993\end{bmatrix}\end{array}.$$

Also its "hard" version, which maps a vector to a one-hot vector where the maximum element becomes 1 and all other elements become 0 (breaking ties arbitrarily):

$$\underset{\mathsf{foo}}{\mathrm{argmax}}\, A = \mathsf{foo}\begin{array}{c}\mathsf{bar}\\\begin{bmatrix}1 & 0 & 0\\0 & 1 & 1\end{bmatrix}\end{array}.$$

Concatenation combines two vectors into one:

$$A \underset{\mathsf{foo}}{\oplus} B = \mathsf{foo}\begin{array}{c}\mathsf{bar}\\\begin{bmatrix}3 & 1 & 4\\1 & 5 & 9\\2 & 7 & 1\\8 & 2 & 8\end{bmatrix}\end{array}$$

$$A \underset{\mathsf{bar}}{\oplus} B = \mathsf{foo}\begin{array}{c}\mathsf{bar}\\\begin{bmatrix}3 & 1 & 4 & 2 & 7 & 1\\1 & 5 & 9 & 8 & 2 & 8\end{bmatrix}\end{array}$$

3.1.5 Renaming and reshaping

It's also very handy to have a function that renames an axis:

$$[A]_{\mathsf{bar}\to\mathsf{baz}} = \mathsf{foo}\begin{array}{c}\mathsf{baz}\\\begin{bmatrix}3 & 1 & 4\\1 & 5 & 9\end{bmatrix}\end{array}$$

We can reshape two or more axes into one axis:

$$[A]_{(\mathsf{foo},\mathsf{bar})\to\mathsf{baz}} = \begin{array}{c}\mathsf{baz}\\\begin{bmatrix}3 & 1 & 4 & 1 & 5 & 9\end{bmatrix}\end{array}$$

Similarly, we can reshape one axis into two or more axes, or even multiple axes into multiple axes. The order of elements in the new axis or axes is undefined. If you need a particular order, you can write a more specific definition.

3.1.6 Matrices

Finally, we briefly consider functions on matrices, for which you have to give *two* axis names (and the order in general matters). Let A be a named tensor with shape $\mathsf{foo}[2] \times \mathsf{bar}[2] \times \mathsf{baz}[2]$:

$$A_{\mathsf{foo}(1)} = \mathsf{bar} \overset{\mathsf{baz}}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}$$

$$A_{\mathsf{foo}(2)} = \mathsf{bar} \overset{\mathsf{baz}}{\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}}$$

$$\det_{\mathsf{bar},\mathsf{baz}} A = \overset{\mathsf{foo}}{\left[\det \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \det \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \right]}$$

$$\det_{\mathsf{baz},\mathsf{bar}} A = \overset{\mathsf{foo}}{\left[\det \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \quad \det \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix} \right]}$$

$$\det_{\mathsf{foo},\mathsf{bar}} A = \overset{\mathsf{baz}}{\left[\det \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} \quad \det \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix} \right]}$$

For matrix inverses, there's no easy way to put a subscript under $\cdot^{-1}$, so we recommend writing $\underset{\mathsf{foo},\mathsf{bar}}{\mathrm{inv}}$.

4 Examples

In this section we give a series of examples illustrating how to use named tensors in various situations, mostly related to machine learning.

4.1 Building blocks

4.1.1 Fully-connected layers

A feedforward neural network looks like this:

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

$$X^1 = \sigma(W^1 \underset{\mathsf{input}}{\cdot} X^0 + b^1) \qquad W^1 \in \mathbb{R}^{\mathsf{hidden1} \times \mathsf{input}} \qquad b^1 \in \mathbb{R}^{\mathsf{hidden1}}$$

$$X^2 = \sigma(W^2 \underset{\mathsf{hidden1}}{\cdot} X^1 + b^2) \qquad W^2 \in \mathbb{R}^{\mathsf{hidden2} \times \mathsf{hidden1}} \qquad b^2 \in \mathbb{R}^{\mathsf{hidden2}}$$

$$X^3 = \sigma(W^3 \underset{\mathsf{hidden2}}{\cdot} X^2 + b^3) \qquad W^3 \in \mathbb{R}^{\mathsf{output} \times \mathsf{hidden2}} \qquad b^3 \in \mathbb{R}^{\mathsf{output}}$$

The layer sizes can be set by writing $|\mathsf{input}| = 100$, etc. Alternatively, we could have called the axes $\mathsf{layer}[n_0]$, $\mathsf{layer}[n_1]$, etc. and set the n_l.

If you don't like repeating the equations for fully-connected layers, you can put them inside a function:

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

where

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

Now $\mathrm{FullConn}^l$ encapsulates both the equation for layer l as well as its parameters (analogous to what TensorFlow and PyTorch call *modules*).

Then the network can be defined like this:

$$X^0 \in \mathbb{R}^{\mathsf{layer}[n_0]}$$
$$X^1 = \mathrm{FullConn}^1(X^0)$$
$$X^2 = \mathrm{FullConn}^2(X^1)$$
$$X^3 = \mathrm{FullConn}^3(X^2).$$

4.1.2 Recurrent neural networks

As a second example, let's define a simple (Elman) RNN. This is similar to the feedforward network, except that the number of timesteps is variable and they all share parameters.

$$x^t \in \mathbb{R}^{\mathsf{input}} \qquad\qquad t = 1, \ldots, n$$
$$W^{\mathrm{h}} \in \mathbb{R}^{\mathsf{hidden} \times \mathsf{hidden}'} \qquad\qquad |\mathsf{hidden}| = |\mathsf{hidden}'|$$
$$W^{\mathrm{i}} \in \mathbb{R}^{\mathsf{input} \times \mathsf{hidden}'}$$
$$b \in \mathbb{R}^{\mathsf{hidden}'}$$
$$h^0 \in \mathbb{R}^{\mathsf{hidden}}$$
$$h^t = \left[\sigma \left(W^{\mathrm{h}} \underset{\mathsf{hidden}}{\bullet} h^{t-1} + W^{\mathrm{i}} \underset{\mathsf{input}}{\bullet} x^t + b \right) \right]_{\mathsf{hidden}' \to \mathsf{hidden}} \qquad t = 1, \ldots, n$$

Here the axis name state has to stay the same across time, and the renaming is necessary because our notation doesn't provide a one-step way to apply a linear transformation (W^{h}) to one axis and put the result in the same axis. For possible solutions, see §**??**.

4.1.3 Attention

In the introduction (§1), we mentioned some difficulties in interpreting the equation for attention as it's usually written. In our notation, it looks like this:

$$\mathrm{Att}\colon \mathbb{R}^{\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}\times\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}\times\mathsf{val}} \to \mathbb{R}^{\mathsf{val}}$$

$$\mathrm{Att}(Q,K,V) = \operatorname*{softmax}_{\mathsf{seq}} \left(\frac{Q \underset{\mathsf{key}}{\cdot} K}{\sqrt{|\mathsf{key}|}} \right) \underset{\mathsf{seq}}{\cdot} V.$$

Sometimes we need to apply a mask to keep from attending to certain positions.

$$\mathrm{Att}\colon \mathbb{R}^{\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}\times\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}\times\mathsf{val}} \times \mathbb{R}^{\mathsf{seq}} \to \mathbb{R}^{\mathsf{val}}$$

$$\mathrm{Att}(Q,K,V,M) = \operatorname*{softmax}_{\mathsf{seq}} \left(\frac{Q \underset{\mathsf{key}}{\cdot} K}{\sqrt{|\mathsf{key}|}} + M \right) \underset{\mathsf{seq}}{\cdot} V.$$

Models often use attention to compute a sequence of values, not just a single value. If Q has (say) a seq' axis, then the above definition computes a sequence of values along the seq' axis. If Q, K, and V have a head axis for multiple attention heads, then it will compute multi-head attention.

4.1.4 Convolution

A 1-dimensional convolution can be easily written by unrolling a tensor and then applying a standard dot product.

$$\mathrm{conv1d}\colon \mathbb{R}^{\mathsf{channels}\times\mathsf{seq}[n]} \to \mathbb{R}^{\mathsf{seq}[n']}$$

$$\mathrm{conv1d}(X;W,b) = W \underset{\mathsf{channels},\mathsf{kernel}}{\cdot} U + b$$

where

$$n' = n - |\mathsf{kernel}| + 1$$
$$W \in \mathbb{R}^{\mathsf{channels}\times\mathsf{kernel}}$$
$$U \in \mathbb{R}^{\mathsf{channels}\times\mathsf{seq}[n']\times\mathsf{kernel}}$$
$$U_{\mathsf{seq}(i),\mathsf{kernel}(j)} = X_{\mathsf{seq}(i+j-1)}$$
$$b \in \mathbb{R}.$$

A 2-dimensional convolution:

$$\mathrm{conv2d}\colon \mathbb{R}^{\mathsf{channels}\times\mathsf{height}[h]\times\mathsf{width}[w]} \to \mathbb{R}^{\mathsf{height}[h']\times\mathsf{width}[w']}$$

$$\mathrm{conv2d}(X;W,b) = W \underset{\mathsf{channels},\mathsf{kh},\mathsf{kw}}{\cdot} U + b$$

where

$$h' = h - |\mathsf{kh}| + 1$$
$$w' = w - |\mathsf{kw}| + 1$$
$$W \in \mathbb{R}^{\mathsf{channels} \times \mathsf{kh} \times \mathsf{kw}}$$
$$U \in \mathbb{R}^{\mathsf{channels} \times \mathsf{height}[h'] \times \mathsf{width}[w'] \times \mathsf{kh} \times \mathsf{kw}}$$
$$U_{\mathsf{height}(i),\mathsf{width}(j),\mathsf{kh}(ki),\mathsf{kw}(kj)} = X_{\mathsf{height}(i+ki-1),\mathsf{width}(j+kj-1)}$$
$$b \in \mathbb{R}.$$

4.1.5   Max pooling

$$\mathrm{maxpool2d}_{kh,kw} \colon \mathbb{R}^{\mathsf{height}[h] \times \mathsf{width}[w]} \to \mathbb{R}^{\mathsf{height}[h/kh] \times \mathsf{width}[w/kw]}$$
$$\mathrm{maxpool2d}_{kh,hw}(X) = \max_{\mathsf{kh},\mathsf{kw}} U$$

where

$$U \in \mathbb{R}^{\mathsf{height}[h/kh] \times \mathsf{width}[w/kw] \times \mathsf{kh}[kh] \times \mathsf{kw}[kw]}$$
$$U_{\mathsf{height}(i),\mathsf{width}(j),\mathsf{kh}(di),\mathsf{kw}(dj)} = X_{\mathsf{height}(i \times kh+di-1),\mathsf{width}(j \times kw+dj-1)}.$$

4.1.6   Normalization layers

Batch, instance, and layer normalization are often informally described using the same equation, but they each correspond to very different functions. They differ by which axes are normalized.

We can define a single generic normalization layer:

$$\mathrm{xnorm}_{\mathsf{ax}} \colon \mathbb{R}^{\mathsf{ax}} \to \mathbb{R}^{\mathsf{ax}}$$
$$\mathrm{xnorm}_{\mathsf{ax}}(X; \gamma, \beta, \epsilon) = \frac{X - \mathrm{mean}_{\mathsf{ax}}(X)}{\sqrt{\mathrm{var}_{\mathsf{ax}}(X) + \epsilon}} \odot \gamma + \beta$$

where

$$\gamma, \beta \in \mathbb{R}^{\mathsf{ax}}$$
$$\epsilon > 0.$$

Now, suppose that the input has three axes:

$$X \in \mathbb{R}^{\mathsf{batch} \times \mathsf{channels} \times \mathsf{layer}}$$

Then the three kinds of normalization layers can be written as:

$$Y = \mathrm{xnorm}_{\mathsf{batch}}(X; \gamma, \beta) \qquad \text{batch normalization}$$
$$Y = \mathrm{xnorm}_{\mathsf{layer}}(X; \gamma, \beta) \qquad \text{instance normalization}$$
$$Y = \mathrm{xnorm}_{\mathsf{layer,channels}}(X; \gamma, \beta) \qquad \text{layer normalization}$$

4.2 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:

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

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

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

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

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

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

where LayerNorm, SelfAtt and FFN are defined below.

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

$$\mathrm{LayerNorm}^l : \mathbb{R}^{\mathsf{layer}} \to \mathbb{R}^{\mathsf{layer}}$$
$$\mathrm{LayerNorm}^l(X) = \underset{\mathsf{layer}}{\mathrm{xnorm}}(X; \beta^l, \gamma^l).$$

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

$$\mathrm{SelfAtt}^l : \mathbb{R}^{\mathsf{seq} \times \mathsf{layer}} \to \mathbb{R}^{\mathsf{seq} \times \mathsf{layer}}$$
$$\mathrm{SelfAtt}^l(X) = Y$$

where

$$|\mathsf{seq}| = |\mathsf{seq}'|$$
$$|\mathsf{key}| = |\mathsf{val}| = |\mathsf{layer}|/|\mathsf{heads}|$$

$$Q = \left[W^{l,Q} \underset{\mathsf{layer}}{\cdot} X\right]_{\mathsf{seq}\to\mathsf{seq}'} \qquad\qquad W^{l,Q} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{layer}\times\mathsf{key}}$$

$$K = W^{l,K} \underset{\mathsf{layer}}{\cdot} X \qquad\qquad W^{l,K} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{layer}\times\mathsf{key}}$$

$$V = W^{l,V} \underset{\mathsf{layer}}{\cdot} X \qquad\qquad W^{l,V} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{layer}\times\mathsf{val}}$$

$$M \in \mathbb{R}^{\mathsf{seq}\times\mathsf{seq}'}$$

$$M_{\mathsf{seq}(i),\mathsf{seq}'(j)} = \begin{cases} 0 & i \le j \\ -\infty & \text{otherwise} \end{cases}$$

$$Y = \sum_{\mathsf{heads}} W^{l,O} \underset{\mathsf{val}}{\cdot} \left[\mathrm{Att}(Q,K,V,M)\right]_{\mathsf{seq}'\to\mathsf{seq}} \quad W^{l,O} \in \mathbb{R}^{\mathsf{heads}\times\mathsf{val}\times\mathsf{layer}}$$

Feedforward neural networks:

$$\mathrm{FFN}^l \colon \mathbb{R}^{\mathsf{layer}} \to \mathbb{R}^{\mathsf{layer}}$$
$$\mathrm{FFN}^l(X) = X^2$$

where

$$X^1 = \mathrm{relu}(W^{l,1} \underset{\mathsf{layer}}{\cdot} X + b^{l,1}) \qquad W^{l,1} \in \mathbb{R}^{\mathsf{hidden}\times\mathsf{layer}} \qquad b^{l,1} \in \mathbb{R}^{\mathsf{hidden}}$$

$$X^2 = \mathrm{relu}(W^{l,2} \underset{\mathsf{hidden}}{\cdot} X^1 + b^{l,2}) \qquad W^{l,2} \in \mathbb{R}^{\mathsf{layer}\times\mathsf{hidden}} \qquad b^{l,2} \in \mathbb{R}^{\mathsf{hidden}}.$$

4.3 LeNet

$$X^0 \in \mathbb{R}^{\mathsf{batch}\times\mathsf{channels}[c_0]\times\mathsf{height}\times\mathsf{width}}$$
$$T^1 = \mathrm{relu}(\mathrm{conv}^1(X^0))$$
$$X^1 = \mathrm{maxpool}^1(T^1)$$
$$T^2 = \mathrm{relu}(\mathrm{conv}^2(X^1))$$
$$X^2 = \left[\mathrm{maxpool}^2(T^2)\right]_{(\mathsf{height},\mathsf{width},\mathsf{channels})\to\mathsf{layer}}$$
$$X^3 = \mathrm{relu}(W^3 \underset{\mathsf{layer}}{\cdot} X^2 + b^3) \qquad\qquad W^3 \in \mathbb{R}^{\mathsf{hidden}\times\mathsf{layer}} \qquad b^3 \in \mathbb{R}^{\mathsf{hidden}}$$
$$O = \underset{\mathsf{classes}}{\mathrm{softmax}}(W^4 \underset{\mathsf{hidden}}{\cdot} X^3 + b^4) \qquad\qquad W^4 \in \mathbb{R}^{\mathsf{classes}\times\mathsf{hidden}} \qquad b^4 \in \mathbb{R}^{\mathsf{classes}}$$

The flattening operation in the equation for X^2 is defined in §**??**. Alternatively, we could have written

$$X^2 = \mathrm{maxpool}^2(T^2)$$
$$X^3 = \mathrm{relu}(W^3 \underset{\mathsf{height},\mathsf{width},\mathsf{channels}}{\cdot} X^2 + b^3) \quad W^3 \in \mathbb{R}^{\mathsf{hidden}\times\mathsf{height}\times\mathsf{width}\times\mathsf{channels}} \quad b^3 \in \mathbb{R}^{\mathsf{hidden}}$$

The convolution and pooling operations are defined as follows:

$$\mathrm{conv}^l(X) = \big[\mathrm{conv2d}(X; W^l, b^l)\big]_{\mathsf{channels}' \to \mathsf{channels}}$$

where

$$|\mathsf{channels}| = |\mathsf{channels}'|$$
$$W^l \in \mathbb{R}^{\mathsf{channels}'[c_l] \times \mathsf{channels}[c_{l-1}] \times \mathsf{kh}[kh_l] \times \mathsf{kw}[kw_l]}$$
$$b^l \in \mathbb{R}^{\mathsf{channels}'[c_l]}$$

and

$$\mathrm{maxpool}^l(X) = \mathrm{maxpool2d}_{ph^l, ph^l}(X).$$

4.4 Other examples

4.4.1 Discrete random variables

Named axes are very helpful for working with discrete random variables, because each random variable can be represented by an axis with the same name. For instance, if A and B are random variables, we can treat $p(\mathsf{B} \mid \mathsf{A})$ and $p(\mathsf{A})$ as tensors:

$$p(\mathsf{B} \mid \mathsf{A}) \in [0,1]^{\mathsf{A} \times \mathsf{B}} \qquad\qquad \sum_{\mathsf{B}} p(\mathsf{B} \mid \mathsf{A}) = 1$$

$$(\mathsf{A}) \in [0,1]^{\mathsf{A}} \qquad\qquad \sum_{\mathsf{A}} p(\mathsf{A}) = 1$$

Then Bayes' rule is just:

$$p(\mathsf{A} \mid \mathsf{B}) = \frac{p(\mathsf{B} \mid \mathsf{A}) \odot p(\mathsf{A})}{p(\mathsf{B} \mid \mathsf{A}) \underset{\mathsf{A}}{\cdot} p(\mathsf{A})}.$$

4.4.2 Continuous bag of words

A continuous bag-of-words model classifies by summing up the embeddings of a sequence of words X and then projecting them to the space of classes.

$$\mathrm{cbow} \colon \{0,1\}^{\mathsf{seq} \times \mathsf{vocab}} \to \mathbb{R}^{\mathsf{seq} \times \mathsf{classes}}$$
$$\mathrm{cbow}(X; E, W) = \mathrm{softmax}(W \underset{\mathsf{class}}{\cdot} E \underset{\mathsf{vocab}}{\cdot} X)$$

where

$$\sum_{\mathsf{vocab}} X = 1$$
$$E \in \mathbb{R}^{\mathsf{vocab} \times \mathsf{hidden}}$$
$$W \in \mathbb{R}^{\mathsf{classes} \times \mathsf{hidden}}.$$

Here, the two contractions can be done in either order, so we leave the parentheses off.

4.4.3 Sudoku ILP

Sudoku puzzles can be represented as binary tiled tensors. Given a grid we can check that it is valid by converting it to a grid of grids. Constraints then ensure that there is one digit per row, per column and per sub-box.

$$\text{check} \colon \{0,1\}^{\mathsf{height}[9] \times \mathsf{width}[9] \times \mathsf{assign}[9]} \to \{0,1\}$$

$$\text{check}(X) = \mathbb{I}\left[\begin{array}{c} \displaystyle\sum_{\mathsf{assign}} Y = 1 \wedge \sum_{\mathsf{height,width}} Y = 1 \wedge \\ \displaystyle\sum_{\mathsf{Height,height}} Y = 1 \wedge \sum_{\mathsf{Width,width}} Y = 1 \end{array} \right]$$

where

$$Y \in \{0,1\}^{\mathsf{Height}[3] \times \mathsf{Width}[3] \times \mathsf{height}[3] \times \mathsf{width}[3] \times \mathsf{assign}[9]}$$

$$Y_{\mathsf{Height}(r),\mathsf{height}(r'),\mathsf{Width}(c),\mathsf{width}(c')} = X_{\mathsf{height}(r\times 3+r'-1),\mathsf{width}(c\times 3+c'-1)}.$$

4.4.4 K-means clustering

The following equations define one step of k-means clustering. Given a set of points X and an initial set of cluster centers C,

$$X \in \mathbb{R}^{\mathsf{batch} \times \mathsf{space}}$$

$$C \in \mathbb{R}^{\mathsf{clusters} \times \mathsf{space}}$$

we repeat the following update: Compute cluster assignments

$$Q = \operatorname*{argmin}_{\mathsf{clusters}} \operatorname*{norm}_{\mathsf{space}}(C - X)$$

then recompute the cluster centers:

$$C \leftarrow \sum_{\mathsf{batch}} \frac{Q \odot X}{Q}.$$

4.4.5 Beam search

Beam search is a commonly used approach for approximate discrete search. Here H is the score of each element in the beam, S is the state of each element in the beam, and f is an update function that returns the score of each state transition.

$$H \in \mathbb{R}^{\mathsf{beam}}$$

$$S \in \{0,1\}^{\mathsf{beam} \times \mathsf{state}} \qquad \sum_{\mathsf{state}} S = 1$$

$$f \colon \{0,1\}^{\mathsf{state}} \to \mathbb{R}^{\mathsf{state}}$$

Then we repeat the following update:

$$H' = \max_{\text{beam}}(H \odot f(S))$$

$$H \leftarrow \max_{\text{state,beam}} H'$$

$$S \leftarrow \operatorname{argmaxk}_{\text{state,beam}} H'$$

where

$$\operatorname*{maxk}_{\text{ax,k}} \colon \mathbb{R}^{\text{ax}} \to \mathbb{R}^{\text{k}}$$

$$\operatorname*{argmaxk}_{\text{ax,k}} \colon \mathbb{R}^{\text{ax}} \to \{0,1\}^{\text{ax,k}}$$

are defined such that $[\operatorname*{maxk}_{\text{ax,k}} A]_{\text{k}(i)}$ is the i-th largest value along axis ax and $A \underset{\text{ax}}{\cdot} (\operatorname*{argmaxk}_{\text{ax,k}} A) = \max_{\text{ax,k}} A$.

We can add a batch axis to H and S and the above equations will work unchanged.

4.4.6   Multivariate normal distribution

In our notation, the application of a bilinear form is more verbose than the standard notation $((X - \mu)^\top \Sigma^{-1}(X - \mu))$, but also makes it look more like a function of two arguments (and would generalize to three or more arguments).

$$\mathcal{N} \colon \mathbb{R}^{\text{d}} \to \mathbb{R}$$

$$\mathcal{N}(X;\mu,\Sigma) = \frac{\exp\left(-\dfrac{1}{2}\left(\operatorname*{inv}_{\text{d1,d2}}(\Sigma) \underset{\text{d1}}{\cdot} [X - \mu]_{\text{d}\to\text{d1}}\right) \underset{\text{d2}}{\cdot} [X - \mu]_{\text{d}\to\text{d2}}\right)}{\sqrt{(2\pi)^{|\text{d}|} \operatorname*{det}_{\text{d1,d2}}(\Sigma)}}$$

where

$$|\text{d}| = |\text{d1}| = |\text{d2}|$$

$$\mu \in \mathbb{R}^{\text{d}}$$

$$\Sigma \in \mathbb{R}^{\text{d1}\times\text{d2}}.$$

5   LATEX Macros

Many of the LaTeX macros used in this document are available in the style file
https://namedtensor.github.io/namedtensor.sty. To use it, put

    \usepackage{namedtensor}

in the preamble of your LaTeX source file (after `\documentclass{article}` but before `\begin{document}`).

The style file contains a small number of macros:

- Use `\name{foo}` to write an axis name: foo.

- Use `A \ndot{foo} B` for contraction: $A \underset{\mathsf{foo}}{\cdot} B$. Similarly, use `A \ncat{foo} B` for concatenation.

- Use `\nsum{foo} A` for summation: $\sum_{\mathsf{foo}} A$.

- Use `\nfun{foo}{qux} A` for a function named qux with a name under it: $\underset{\mathsf{foo}}{\mathrm{qux}} A$.

- Use `\nmov{foo}{bar}{A}` for renaming: $[A]_{\mathsf{foo}\to\mathsf{bar}}$.

6 Formal Definitions

6.1 Axes, shapes, records, and types

A *named index* is a pair (ax, i), written $\mathsf{ax}(i)$, where ax is a *name* and i is usually a natural number. An *axis* is a set of named indices $\{(\mathsf{ax}, i) | i \in I\}$ all sharing the same name. We deal with axes of the form $\mathsf{ax}[[n]]$ (that is, $\mathsf{ax}[\{1, \ldots, n\}]$) so frequently that we abbreviate this as $\mathsf{ax}[n]$. We sometimes use ax to specify the abstract axis, without specifying the set of its support, when the size can be inferred from context or when the particular size is immaterial (for example when defining a polymorphic function that can take tensors with given axes regardless of their size).

A *type* is a set of (distinct) axes, without specifying their support size. A *shape* is a set of axes with the support specified. For example, $\{\mathsf{foo}, \mathsf{bar}\}$ is a type and $\{\mathsf{foo}[10], \mathsf{bar}[20]\}$ is a shape. If $\mathcal{S} = \{\mathsf{ax}^1, \ldots, \mathsf{ax}^d\}$ is a shape then a *record* of shape $\mathcal{S}$ is a set $\{i_1, \ldots, i_d\}$ where $i_j \in \mathsf{ax}^j$. We denote by $\mathrm{rec}\,\mathcal{S}$ the set of all records of shape $\mathcal{S}$. A *tensor* of shape $\mathcal{S}$ is a function mapping all records of shape $\mathcal{S}$ into the real numbers. We write $A \in \mathbb{R}^{\mathsf{ax}^1 \times \cdots \times \mathsf{ax}^d}$ for the set of all tensors of the shape $\{\mathsf{ax}^1, \ldots, \mathsf{ax}^d\}$. If A is a tensor of shape $\mathcal{S}$ and $r \in \mathrm{rec}\,\mathcal{S}$ then we denote by A_r the real numbered map into by r.

A *partial record* of shape $\mathcal{S} = \{\mathsf{ax}^1, \ldots, \mathsf{ax}^d\}$ is a set $\{1_1, \ldots, i_{d'}\}$ such that $d' \leq d$ and each of the i_j's is a member of a distinct element of $\mathcal{S}$. If $r = \{i_1, \ldots, i_{d'}\}$ is a partial record of shape $\mathcal{S}$, then we denote by $\mathcal{S}(r)$ the set $\{\mathsf{ax}^1, \ldots \mathsf{ax}^{d'}\} \subseteq \mathcal{S}$ such that $i_j \in \mathsf{ax}^j$ for $j = 1, \ldots, d'$. If A is a tensor of shape $\mathcal{S} = \{\mathsf{ax}^1, \ldots, \mathsf{ax}^d\}$ and $r = \{i_1, \ldots, i_{d'}\}$ is a partial record of shape $\mathcal{S}$, then A_r is the tensor of shape $\mathcal{S} \setminus \mathcal{S}(r)$ that maps every record $r' = \{j_1, \ldots, j_{d-d'}\}$ to the value $A_{r \cup r'}$.

6.2 Extending functions to named tensors

In §1, we described several classes of functions that can be extended to named tensors. Here, we define how to do this for general functions.

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.

Let $f \colon F^{\mathcal{S}} \to G^{\mathcal{T}}$ be a function from tensors to tensors. For any shape $\mathcal{U}$ orthogonal to both $\mathcal{S}$ and $\mathcal{T}$, we can extend f to:

$$f : F^{\mathcal{S} \cup \mathcal{U}} \to G^{\mathcal{T} \cup \mathcal{U}}$$

$$[f(A)]_u = f(A_u) \qquad \text{for all } u \in \text{rec}\,\mathcal{U}.$$

If f is a multary function, we can extend its arguments to larger shapes, and we don't have to extend all the arguments with the same names. We consider just the case of two arguments; three or more arguments are analogous. 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 $\mathcal{U}' = \mathcal{S}' \cup \mathcal{T}'$ is orthogonal to $\mathcal{U}$, we can extend f to:

$$f : F^{\mathcal{S} \cup \mathcal{S}'} \times G^{\mathcal{T} \cup \mathcal{T}'} \to H^{\mathcal{U} \cup \mathcal{U}'}$$

$$[f(A, B)]_u = f\left(A_{u|_{\mathcal{S}'}}, B_{u|_{\mathcal{T}'}}\right) \qquad \text{for all } u \in \text{rec}\,\mathcal{U}'.$$

All of the tensor operations described in §**??** can be defined in this way. For example, the contraction operator extends the following "named dot-product":

$$\underset{\mathsf{ax}}{\cdot} : F^{\mathsf{ax}[n]} \times F^{\mathsf{ax}[n]} \to F$$

$$A \underset{\mathsf{ax}}{\cdot} B = \sum_{i=1}^{n} A_{\mathsf{ax}(i)} B_{\mathsf{ax}(i)}.$$

7 Extensions

7.1 Index types

We have defined an axis as a pair $\mathsf{ax}[I]$, where ax is a name and I is a set, usually $[n]$ for some n. In this section, we consider some other possibilities for I.

7.1.1 Non-integral types

The sets I don't have to contain integers. For example, if V is the vocabulary of a natural language ($V = \{\mathsf{cat}, \mathsf{dog}, \ldots\}$), we could define a matrix of word embeddings:

$$E \in \mathbb{R}^{\mathsf{vocab}[V] \times \mathsf{emb}[d]}.$$

7.1.2 Integers with units

If u is a symbol and $n > 0$, define $[n]\mathsf{u} = \{1\mathsf{u}, 2\mathsf{u}, \ldots, n\mathsf{u}\}$. You could think of u as analogous to a physical unit, like kilograms. The elements of $[n]\mathsf{u}$ can be added and subtracted like integers ($a\mathsf{u} + b\mathsf{u} = (a + b)\mathsf{u}$) or multiplied by unitless integers ($c \cdot a\mathsf{u} = (c \cdot a)\mathsf{u}$), but numbers with different units are different ($a\mathsf{u} \neq a\mathsf{v}$).

Then the set $[n]\mathsf{u}$ could be used as an index set, which would prevent the axis from being aligned with another axis that uses different units. For example, if we want to define a tensor representing an image, we might write

$$A \in \mathbb{R}^{\mathsf{height}[[h]\mathsf{pixels}] \times \mathsf{width}[[w]\mathsf{pixels}]}.$$

If we have another tensor representing a go board, we might write

$$B \in \mathbb{R}^{\mathsf{height}[[n]\mathsf{points}] \times \mathsf{width}[[n]\mathsf{points}]},$$

and even if it happens that $h = w = n$, it would be incorrect to write $A + B$ because the units do not match.

7.1.3 Tuples of integers

An index set could also be $[m] \times [n]$, which would be a way of sneaking ordered indices into named tensors, useful for matrix operations. For example, instead of defining an inv operator that takes two subscripts, we could write

$$A \in \mathbb{R}^{\mathsf{ax}[m \times n]} = \mathbb{R}^{\mathsf{ax}[[m] \times [n]]}$$
$$B = \operatorname*{inv}_{\mathsf{ax}} A.$$

We could also define an operator $\circ$ for matrix-matrix and matrix-vector multiplication:

$$c \in \mathbb{R}^{\mathsf{ax}[n]}$$
$$D = A \underset{\mathsf{ax}}{\circ} B \underset{\mathsf{ax}}{\circ} c.$$

7.2 Duality

In applied linear algebra, we distinguish between column and row vectors; in pure linear algebra, vector spaces and dual vector spaces; in tensor algebra, contravariant and covariant indices; in quantum mechanics, bras and kets. Do we need something like this?

In §?? we saw that defining an RNN requires renaming of axes, because a linear transformation must map one axis to another axis; if we want to map an axis to itself, we need to use renaming.

In this section, we describe three possible solutions to this problem, and welcome comments about which (if any) would be best.

7.2.1   Contracting two names

We define a version of the contraction operator that can contract two axes with different names (and the same size):

$$\underset{\mathsf{ax1|ax2}}{\bullet} : F^{\mathsf{ax1}[n]} \times F^{\mathsf{ax2}[n]} \to F$$

$$A \underset{\mathsf{ax1|ax2}}{\bullet} B = \sum_{i=1}^{n} A_{\mathsf{ax1}(i)} B_{\mathsf{ax2}(i)}.$$

For example, the RNN would look like this.

$$x^{(t)} \in \mathbb{R}^{\mathsf{emb}[d]}$$
$$h^{(t)} \in \mathbb{R}^{\mathsf{state}[d]}$$
$$A \in \mathbb{R}^{\mathsf{state}[d] \times \mathsf{state}'[d]}$$
$$B \in \mathbb{R}^{\mathsf{emb}[d] \times \mathsf{state}[d]}$$
$$c \in \mathbb{R}^{\mathsf{state}[d]}$$
$$h^{(t+1)} = \tanh\left(A \underset{\mathsf{state}'|\mathsf{state}}{\bullet} h^{(t)} + B \underset{\mathsf{emb}}{\bullet} x^{(t)} + c \right)$$

7.2.2   Starred axis names

If ax is a name, we also allow a tensor to have an axis ax∗ (alternatively: superscript ax). Multiplication contracts starred axes in the left operand with non-starred axes in the right operand.

$$x^{(t)} \in \mathbb{R}^{\mathsf{emb}[d]}$$
$$h^{(t)} \in \mathbb{R}^{\mathsf{state}[d]}$$
$$A \in \mathbb{R}^{\mathsf{state}*[d] \times \mathsf{state}[d]}$$
$$B \in \mathbb{R}^{\mathsf{emb}*[d] \times \mathsf{state}[d]}$$
$$c \in \mathbb{R}^{\mathsf{state}[d]}$$
$$h^{(t+1)} = \tanh\left(A \underset{\mathsf{state}}{\bullet} h^{(t)} + B \underset{\mathsf{emb}}{\bullet} x^{(t)} + c \right)$$

The contraction operator can be defined as:

$$\underset{\mathsf{ax}}{\bullet} : F^{\mathsf{ax}*[n]} \times F^{\mathsf{ax}[n]} \to F$$

$$A \underset{\mathsf{ax}}{\bullet} B = \sum_{i=1}^{n} A_{\mathsf{ax}*(i)} B_{\mathsf{ax}(i)}.$$

There are a few variants of this idea that have been floated:

1. $\cdot$ (no subscript) contracts every starred axis in its left operand with every corresponding unstarred axis in its right operand. Rejected.

2. $\underset{\mathsf{ax}}{\cdot}$ contracts ax with ax, and we need another notation like $\underset{\mathsf{ax}(*)}{\cdot}$ or $\underset{\mathsf{ax}}{\times}$ for contracting $\mathsf{ax}*$ with ax.

3. $\underset{\mathsf{ax}}{\cdot}$ always contracts $\mathsf{ax}*$ with ax; there's no way to contract ax with ax.

7.2.3 Named and numbered axes

We allow axes to have names that are natural numbers $1, 2, \ldots$, and we define "numbering" and "naming" operators:

$$
\begin{array}{ll}
A_{\mathsf{ax}} & \text{rename axis } \mathsf{ax} \text{ to } 1 \\
A_{\mathsf{ax1},\mathsf{ax2}} & \text{rename axis } \mathsf{ax1} \text{ to } 1 \text{ and } \mathsf{ax2} \text{ to } 2 \\
A_{\to\mathsf{ax}} & \text{rename axis } 1 \text{ to } \mathsf{ax} \\
A_{\to\mathsf{ax1},\mathsf{ax2}} & \text{rename axis } 1 \text{ to } \mathsf{ax1} \text{ and } 2 \text{ to } \mathsf{ax2}
\end{array}
$$

The numbering operators are only defined on tensors that have no numbered axes.

Then we adopt the convention that standard vector/matrix operations operate on the numbered axes. For example, vector dot-product always uses axis 1 of both its operands, so that we can write

$$C = A_{\mathsf{ax}} \cdot B_{\mathsf{ax}}$$

equivalent to $C = A \underset{\mathsf{ax}}{\cdot} B$.

Previously, we had to define a new version of every operation; most of the time, it looked similar to the standard version (e.g., $\max$ vs $\max_{\mathsf{ax}}$), but occasionally it looked quite different (e.g., matrix inversion). With numbered axes, we can use standard notation for everything. (This also suggests a clean way to integrate code that uses named tensors with code that uses ordinary tensors.)

We also get the renaming operation for free: $A_{\mathsf{ax1}\to\mathsf{ax2}} = [A_{\mathsf{ax1}}]_{\to\mathsf{ax2}}$ renames axis $\mathsf{ax1}$ to $\mathsf{ax2}$.

Finally, this notation alleviates the duality problem, as can be seen in the definition of a RNN:

$$
\begin{aligned}
x^{(t)} &\in \mathbb{R}^{\mathsf{emb}[d]} \\
h^{(t)} &\in \mathbb{R}^{\mathsf{state}[d]} \\
A &\in \mathbb{R}^{\mathsf{state}[d]\times\mathsf{state}'[d]} \\
B &\in \mathbb{R}^{\mathsf{state}[d]\times\mathsf{emb}[d]} \\
c &\in \mathbb{R}^{\mathsf{state}[d]} \\
h^{(t+1)}_{\mathsf{state}} &= \tanh\left(A_{\mathsf{state},\mathsf{state}'}\, h^{(t)}_{\mathsf{state}} + B_{\mathsf{state},\mathsf{emb}}\, x^{(t)}_{\mathsf{emb}} + c_{\mathsf{state}}\right)
\end{aligned}
$$

or equivalently,

$$h^{(t+1)} = \tanh\left(A_{\mathsf{state}'} \cdot h^{(t)}_{\mathsf{state}} + B_{\mathsf{emb}} \cdot x^{(t)}_{\mathsf{emb}} + c\right)$$

Attention:

$$\mathrm{Att}\colon \mathbb{R}^{\mathsf{seq}'[n']\times\mathsf{key}[d_k]} \times \mathbb{R}^{\mathsf{seq}[n]\times\mathsf{key}[d_k]} \times \mathbb{R}^{\mathsf{seq}[n]\times\mathsf{val}[d_v]} \to \mathbb{R}^{\mathsf{seq}'[n']\times\mathsf{val}[d_v]}$$

$$\mathrm{Att}(Q,K,V) = \underset{\mathsf{seq}}{\mathrm{softmax}}\left[\frac{Q_{\mathsf{key}} \cdot K_{\mathsf{key}}}{\sqrt{d_k}}\right] \cdot V_{\mathsf{seq}}$$

Multivariate normal distribution:

$$X \in \mathbb{R}^{\mathsf{batch}[b]\times\mathsf{d}[k]}$$
$$\mu \in \mathbb{R}^{\mathsf{d}[k]}$$
$$\Sigma \in \mathbb{R}^{\mathsf{d}[k]\times\mathsf{d}'[k]}$$

$$\mathcal{N}(X;\mu,\Sigma) = \frac{\exp\left(-\frac{1}{2}[X-\mu]_{\mathsf{d}}^{\top}\,\Sigma^{-1}_{\mathsf{d},\mathsf{d}'}\,[X-\mu]_{\mathsf{d}}\right)}{\sqrt{(2\pi)^k\,\det\Sigma_{\mathsf{d},\mathsf{d}'}}}$$

Because this notation can be a little more verbose (often requiring you to write axis names twice), we'd keep around the notation $A \underset{\mathsf{ax}}{\cdot} B$ as a shorthand for $A_{\mathsf{ax}} \cdot B_{\mathsf{ax}}$. We'd also keep named reductions, or at least $\underset{\mathsf{ax}}{\mathrm{softmax}}$.

7.3 Indexing with a tensor of indices

Contributors: Tongfei Chen and Chu-Cheng Lin

NumPy defines two kinds of *advanced* (also known as *fancy*) indexing: by integer arrays and by Boolean arrays. Here, we generalize indexing by integer arrays to named tensors. That is, if A is a named tensor with D indices and $\iota^1,\ldots,\iota^D$ are named tensors, called "indexers," what is $A_{\iota^1,\ldots,\iota^D}$?

Advanced indexing could be derived by taking a function

$$\underset{\mathsf{ax}}{\mathrm{index}}\colon F^{\mathsf{ax}[I]} \times I \to F$$
$$\underset{\mathsf{ax}}{\mathrm{index}}(A,i) = A_{\mathsf{ax}(i)}$$

and extending it to higher-order tensors in its second argument according to the rules in §**??**. But because that's somewhat abstract, we give a more concrete definition below.

We first consider the case where all the indexers have the same shape $\mathcal{S}$:

$$A \in F^{\mathsf{ax}_1[I_1]\times\cdots\times\mathsf{ax}_D[I_D]}$$
$$\iota^d \in I_d^{\mathcal{S}} \qquad\qquad\qquad d = 1,\ldots,D.$$

Then $A_{\iota^1,\ldots,\iota^D}$ is the named tensor with shape $\mathcal{S}$ such that for any $s \in \operatorname{rec}\mathcal{S}$,

$$[A_{\iota^1,\ldots,\iota^D}]_s = A_{\iota^1_s,\ldots,\iota^D_s}.$$

More generally, suppose the indexers have different but compatible shapes:

$$A \in F^{\mathsf{ax}_1[I_1]\times\cdots\times\mathsf{ax}_D[I_D]}$$
$$\iota^d \in I_d^{\mathcal{S}_d} \qquad\qquad\qquad d = 1,\ldots,D,$$

where the $\mathcal{S}_d$ are pairwise compatible. Then $A_{\iota^1,\ldots,\iota^D}$ is the named tensor with shape $\mathcal{S} = \bigcup_d \mathcal{S}_d$ such that for any $s \in \operatorname{rec}\mathcal{S}$,

$$[A_{\iota^1,\ldots,\iota^D}]_s = A_{\iota^1_{s|_{\mathcal{S}_1}},\ldots,\iota^D_{s|_{\mathcal{S}_D}}}.$$

Let's consider a concrete example in natural language processing. Consider a batch of sentences encoded as a sequence of word vectors, that is, a tensor $X \in \mathbb{R}^{\mathsf{batch}[B]\times\mathsf{sent}[N]\times\mathsf{emb}[E]}$. For each sentence, we would like to take out the encodings of a particular span for each sentence $b \in [B]$ in the batch, resulting in a tensor $Y \in \mathbb{R}^{\mathsf{batch}[B]\times\mathsf{span}[M]\times\mathsf{emb}[E]}$.

We create a indexer for the sent axis: $\iota \in [N]^{\mathsf{batch}[B]\times\mathsf{span}[M]}$ that selects the desired tokens. Also define the function

$$\operatorname*{arange}_{\mathsf{ax}}(I) \in I^{\mathsf{ax}[I]}$$
$$\left[\operatorname*{arange}_{\mathsf{ax}}(I)\right]_{\mathsf{ax}(i)} = i$$

which generalizes the NumPy function of the same name.

Then we can write

$$Y = X_{\mathsf{batch}(\iota),\mathsf{sent}(\operatorname*{arange}_{\mathsf{sent}}(n)),\mathsf{emb}(\operatorname*{arange}_{\mathsf{emb}}(E))}.$$

Acknowledgements

References

Tongfei Chen. 2017. Typesafe abstractions for tensor operations. In *Proceedings of the 8th ACM SIGPLAN International Symposium on Scala*, SCALA 2017, pages 45–50.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

Dougal Maclaurin, Alexey Radul, Matthew J. Johnson, and Dimitrios Vytiniotis. 2019. Dex: array programming with typed indices. In *NeurIPS Workshop on Program Transformations for ML*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Alexander Rush. 2019. Named tensors. Open-source software.

Nishant Sinha. 2018. Tensor shape (annotation) library. Open-source software.

Torch Contributors. 2019. Named tensors. PyTorch documentation.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.