Named Tensor Notation

David Chiang
University of Notre Dame

Sasha Rush
Cornell University

Boaz Barak
Harvard University

Version 0.2

Contents

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 less well-suited to talking about neural networks. Consider the following equation (Vaswani et al., 2017):

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

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, the notation becomes more complicated and leaves more questions unanswered. With multiple sentences in a minibatch, the notation becomes more complicated still, and most papers wisely leave this detail out.

Libraries for programming with neural networks (Harris et al., 2020; Paszke et al., 2019) provide multidimensional arrays, called tensors (although usually without the theory associated with tensors in linear algebra and physics), and a rich array of operations on tensors. But they inherit from math the convention of identifying axes by *position*, making code bug-prone. Quite a few libraries have been developed to identify axes by *name* instead: Nexus (Chen, 2017), tsalib (Sinha, 2018), NamedTensor (Rush, 2019), named tensors in PyTorch (Torch Contributors, 2019), and Dex (Maclaurin et al., 2019).

Back in the realm of mathematical notation, then, we want two things: first, the flexibility of working with multidimensional arrays, and second, the perspicuity of identifying axes by name instead of by position. This document describes our proposal to do both.

As a preview, the above equation becomes

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

making it unambiguous which axis each operation applies to. The same equation works with multiple heads and with minibatching.

More examples of the notation are given in §3.

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.

2 Informal Overview

Let's think first about the usual notions of vectors, matrices, and tensors, without named axes.

Define $[n] = \{1, \ldots, n\}$. We can think of a size-n real vector v as a function from $[n]$ to $\mathbb{R}$. We get the ith element of v by applying v to i, but we normally write this as v_i (instead of $v(i)$).

Similarly, we can think of an $m \times n$ real matrix as a function from $[m] \times [n]$ to $\mathbb{R}$, and an $l \times m \times n$ real tensor as a function from $[l] \times [m] \times [n]$ to $\mathbb{R}$. In general, then, real tensors are functions from *tuples of natural numbers* to reals.

2.1 Named tensors

We want to make tensors into functions, no longer on tuples, but on *records*, which look like this:

$$\{\mathsf{foo}(1), \mathsf{bar}(3)\}$$

where foo and bar are *names* (written in sans-serif font), mapped to 1 and 3, respectively. The pairs $\mathsf{foo}(1)$ and $\mathsf{bar}(3)$ are called *named indices*. Their order doesn't matter: $\{\mathsf{foo}(1), \mathsf{bar}(3)\}$ and $\{\mathsf{bar}(3), \mathsf{foo}(1)\}$ are the same record.

The set of records that can be used to index a named tensor is defined by a *shape*, which looks like this:

$$\mathsf{foo}[2] \times \mathsf{bar}[3]$$

which stands for records where foo's index ranges from 1 to 2, and bar's index ranges from 1 to 3. The pairs $\mathsf{foo}[2]$ and $\mathsf{bar}[3]$ are called *axes*, and again their order doesn't matter: $\mathsf{foo}[2] \times \mathsf{bar}[3]$ and $\mathsf{bar}[3] \times \mathsf{foo}[2]$ are the same shape.

Then, a real *named tensor* is a function from (the set of records defined by) a shape to the real numbers. For example, here is a tensor with shape $\mathsf{foo}[2] \times \mathsf{bar}[3]$.

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

The same tensor can also be written

$$A = \begin{array}{c} \\ \mathsf{bar} \end{array} \begin{array}{c} \begin{array}{cc} \multicolumn{2}{c}{\mathsf{foo}} \end{array} \\ \begin{bmatrix} 3 & 1 \\ 1 & 5 \\ 4 & 9 \end{bmatrix} \end{array}.$$

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

$$A_{\mathsf{foo}(1)} = \begin{array}{c} \begin{array}{ccc} \multicolumn{3}{c}{\mathsf{bar}} \end{array} \\ \begin{bmatrix} 3 & 1 & 4 \end{bmatrix} \end{array} \qquad A_{\mathsf{bar}(3)} = \begin{array}{c} \begin{array}{cc} \multicolumn{2}{c}{\mathsf{foo}} \end{array} \\ \begin{bmatrix} 4 & 9 \end{bmatrix} \end{array}.$$

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 §3 for more examples.

2.2 Named tensor operations

2.2.1 Elementwise operations

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

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

More elementwise unary operations:

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

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

$$B = \begin{array}{c} \\ \mathsf{foo} \end{array} \begin{array}{c} \begin{array}{ccc} \multicolumn{3}{c}{\mathsf{bar}} \end{array} \\ \begin{bmatrix} 2 & 7 & 1 \\ 8 & 2 & 8 \end{bmatrix} \end{array}$$

then

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

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.

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

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

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

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:

$A + B$	addition
$A - B$	subtraction
$A \odot B$	elementwise (Hadamard) product
$\frac{A}{B}$	elementwise division
$\max\{A, B\}$	elementwise maximum
$\min\{A, B\}$	elementwise minimum

2.2.2 Reductions

The same rules for alignment and broadcasting apply to functions that take tensor as arguments or return tensors. The gory details are in §5.3, 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, we can sum over

the foo axis like this:

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

and sum over the bar axis like this:

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

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

$$\sum_{\mathsf{foo}} A = \sum_{i=1}^{n} A_{\mathsf{foo}(i)} = \begin{bmatrix} 4 & 6 & 13 \end{bmatrix}^{\mathsf{bar}}$$

$$\min_{\mathsf{foo}} A = \min\{A_{\mathsf{foo}(i)} \mid 1 \le i \le n\} = \begin{bmatrix} 1 & 1 & 4 \end{bmatrix}^{\mathsf{bar}}$$

$$\max_{\mathsf{foo}} A = \max\{A_{\mathsf{foo}(i)} \mid 1 \le i \le n\} = \begin{bmatrix} 3 & 5 & 9 \end{bmatrix}^{\mathsf{bar}}$$

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

$$\operatorname*{mean}_{\mathsf{foo}} A = \frac{1}{n} \sum_{\mathsf{foo}} A = \begin{bmatrix} 2 & 3 & 6.5 \end{bmatrix}^{\mathsf{bar}}$$

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

The summation, max, and min operators are overloaded. If the operator is applied to a tensor and has an axis under it, then it's a reduction performed over the axis. But if it is applied to a set of tensors and has no axis under it, then it's an elementwise operation performed over the set. (For example, in the first equation above, $\sum_{\mathsf{foo}} A$ is a reduction over axis foo, while $\sum_{i=1}^{n} A_{\mathsf{foo}(i)}$ is an elementwise sum over the set $\{A_{\mathsf{foo}(i)} \mid i = 1, \ldots, n\}$.)

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

$$\sum_{\mathsf{foo},\mathsf{bar}} A = \sum_i \sum_j A_{\mathsf{foo}(i),\mathsf{bar}(j)} = 23.$$

But unlike in NumPy and its derivatives, a reduction with no axis under it is performed over zero axes (which in most cases does nothing).

2.2.3 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. You can think of this operator as elementwise multiplication followed by summing over an axis. It can be used, for example, for matrix multiplication:

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

$$A \underset{\mathsf{bar}}{\odot} C = \sum_i A_{\mathsf{bar}(i)} C_{\mathsf{bar}(i)} = \begin{array}{c} \\ \mathsf{foo} \end{array} \begin{array}{c} \mathsf{baz} \\ \begin{bmatrix} 17 & -17 \\ 53 & -53 \end{bmatrix} \end{array}$$

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

You can also write multiple names to contract multiple axes at once. An operator $\odot$ with no axis name under it contracts zero axes and is equivalent to elementwise multiplication (which is why we use the same symbol for both).

Finally, we define a version of the contraction operator that can contract two axes with different names (and the same size). The most common use case for this is applying a linear transformation from an axis to itself:

$$T = \begin{array}{c} \\ \mathsf{foo} \end{array} \begin{array}{c} \mathsf{foo}' \\ \begin{bmatrix} 10 & 0 \\ 0 & 100 \end{bmatrix} \end{array}$$

$$T \underset{\mathsf{foo}'|\mathsf{foo}}{\odot} A = \sum_i T_{\mathsf{foo}'(i)} A_{\mathsf{foo}(i)} = \begin{array}{c} \\ \mathsf{foo} \end{array} \begin{array}{c} \mathsf{bar} \\ \begin{bmatrix} 30 & 10 & 40 \\ 100 & 500 & 900 \end{bmatrix} \end{array}.$$

The prime on foo' acts kind of like a transpose, but you can use whatever naming convention you want (we also considered foo^*).

2.2.4 Vectors to vectors

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

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

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):

$$\underset{\mathsf{foo}}{\operatorname{argmax}} A = \begin{matrix} \\ \mathsf{foo} \end{matrix} \begin{matrix} \mathsf{bar} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \end{matrix}.$$

Concatenation combines two vectors into one:

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

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

2.2.5 Renaming and reshaping

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

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

We can reshape two or more axes into one axis:

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

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.

2.2.6 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 $\mathsf{foo}[2] \times \mathsf{bar}[2] \times \mathsf{baz}[2]$:

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

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

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

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

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

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

3 Examples

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

3.1 Building blocks

3.1.1 Fully-connected layers

A feedforward neural network looks like this:

$$\begin{aligned}
&X^0 \in \mathbb{R}^{\mathsf{input}} \\
&X^1 = \sigma(W^1 \underset{\mathsf{input}}{\odot} X^0 + b^1) && W^1 \in \mathbb{R}^{\mathsf{hidden1} \times \mathsf{input}} && b^1 \in \mathbb{R}^{\mathsf{hidden1}} \\
&X^2 = \sigma(W^2 \underset{\mathsf{hidden1}}{\odot} X^1 + b^2) && W^2 \in \mathbb{R}^{\mathsf{hidden2} \times \mathsf{hidden1}} && b^2 \in \mathbb{R}^{\mathsf{hidden2}} \\
&X^3 = \sigma(W^3 \underset{\mathsf{hidden2}}{\odot} X^2 + b^3) && W^3 \in \mathbb{R}^{\mathsf{output} \times \mathsf{hidden2}} && b^3 \in \mathbb{R}^{\mathsf{output}}
\end{aligned}$$

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:

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

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 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:

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

3.1.2 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.

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

3.1.3 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:

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

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

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

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 heads axis for multiple attention heads, then it will compute multi-head attention.

3.1.4 Convolution

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

$$\begin{aligned}\text{conv1d}\colon \mathbb{R}^{\mathsf{channels}\times\mathsf{seq}[n]} &\to \mathbb{R}^{\mathsf{seq}[n-|\mathsf{kernel}|+1]}\\ \text{conv1d}(X;W,b) &= W \underset{\mathsf{channels}'|\mathsf{channels},\mathsf{kernel}}{\odot} U + b\end{aligned}$$

where

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

A 2-dimensional convolution:

$$\begin{aligned}\text{conv2d}\colon \mathbb{R}^{\mathsf{channels}\times\mathsf{height}[h]\times\mathsf{width}[w]} &\to \mathbb{R}^{\mathsf{height}[h-|\mathsf{kh}|+1]\times\mathsf{width}[w-|\mathsf{kw}|+1]}\\ \text{conv2d}(X;W,b) &= W \underset{\mathsf{channels}'|\mathsf{channels},\mathsf{kh},\mathsf{kw}}{\odot} U + b\end{aligned}$$

where

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

3.1.5 Max pooling

$$\begin{aligned}\text{maxpool1d}_k\colon \mathbb{R}^{\mathsf{seq}[n]} &\to \mathbb{R}^{\mathsf{seq}[n/k]}\\ \text{maxpool1d}_k(X) &= \max_{\mathsf{k}} U\end{aligned}$$

where

$$\begin{aligned}U &\in \mathbb{R}^{\mathsf{seq}[n/k]\times\mathsf{k}[k]}\\ U_{\mathsf{seq}(i),\mathsf{k}(di)} &= X_{\mathsf{seq}(i\times k+di-1)}.\end{aligned}$$

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

where

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

3.1.6 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:

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

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:

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

3.2 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:

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

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

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

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

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

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

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 §3.1.3; 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

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

Feedforward neural networks:

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

where

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

3.3 LeNet

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

The flattening operation in the equation for X^2 is defined in §2.2.5. Alternatively, we could have written

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

The convolution and pooling operations are defined as follows:

$$\mathrm{conv}^l(X) = \mathrm{conv2d}(X; W^l, b^l)$$

where

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

and

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

3.4 Other examples

3.4.1 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:

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

Then many common operations on probability distributions can be expressed in terms of tensor operations:

$$p(\mathsf{A}, \mathsf{B}) = p(\mathsf{B} \mid \mathsf{A}) \odot p(\mathsf{A}) \qquad \text{chain rule}$$
$$p(\mathsf{B}) = \sum_{\mathsf{A}} p(\mathsf{A}, \mathsf{B}) = p(\mathsf{B} \mid \mathsf{A}) \underset{\mathsf{A}}{\odot} p(\mathsf{A}) \qquad \text{marginalization}$$
$$p(\mathsf{A} \mid \mathsf{B}) = \frac{p(\mathsf{A}, \mathsf{B})}{p(\mathsf{B})} = \frac{p(\mathsf{B} \mid \mathsf{A}) \odot p(\mathsf{A})}{p(\mathsf{B} \mid \mathsf{A}) \underset{\mathsf{A}}{\odot} p(\mathsf{A})}. \qquad \text{Bayes' rule}$$

3.4.2 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.

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

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.

3.4.3 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.

$$\text{check}\colon \{0,1\}^{\mathsf{height}[9] \times \mathsf{width}[9] \times \mathsf{assign}[9]} \to \{0,1\}$$
$$\text{check}(X) = \mathbb{I}\left[\begin{array}{l} \sum_{\mathsf{assign}} X = 1 \wedge \sum_{\mathsf{height},\mathsf{width}} Y = 1 \wedge \\ \sum_{\mathsf{height}} X = 1 \wedge \sum_{\mathsf{width}} X = 1 \end{array}\right]$$

where

$$Y \in \{0,1\}^{\mathsf{height}'[3] \times \mathsf{width}'[3] \times \mathsf{height}[3] \times \mathsf{width}[3] \times \mathsf{assign}[9]}$$
$$Y_{\mathsf{height}'(h'),\mathsf{height}(h),\mathsf{width}'(w'),\mathsf{width}(w)} = X_{\mathsf{height}(3h'+h-1),\mathsf{width}(3w'+w-1)}.$$

3.4.4 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:

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

3.4.5 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:

$$H' = \max_{\mathsf{beam}}(H \odot f(S))$$
$$H \leftarrow \operatorname*{maxk}_{\mathsf{state},\mathsf{beam}} H'$$
$$S \leftarrow \operatorname*{argmaxk}_{\mathsf{state},\mathsf{beam}} H'$$

where

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

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

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

3.4.6 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).

$$\mathcal{N} \colon \mathbb{R}^{\mathsf{d}} \to \mathbb{R}$$
$$\mathcal{N}(X;\mu,\Sigma) = \frac{\exp\left(-\frac{1}{2}\left(\operatorname*{inv}_{\mathsf{d1},\mathsf{d2}} \Sigma \underset{\mathsf{d1}|\mathsf{d}}{\odot} (X-\mu)\right) \underset{\mathsf{d2}|\mathsf{d}}{\odot} (X-\mu)\right)}{\sqrt{(2\pi)^{|\mathsf{d}|} \operatorname*{det}_{\mathsf{d1},\mathsf{d2}} \Sigma}}$$

where

$$|\mathsf{d}| = |\mathsf{d1}| = |\mathsf{d2}|$$
$$\mu \in \mathbb{R}^{\mathsf{d}}$$
$$\Sigma \in \mathbb{R}^{\mathsf{d1} \times \mathsf{d2}}.$$

4 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:

- Basics
  - Use `\name{foo}` to write an axis name: foo.
  - Use `\mathbb{R}^{\nset{foo}{2}}` to write a set of tensors: $\mathbb{R}^{\mathsf{foo}[2]}$.
  - Use `A_{\nidx{foo}{1}}` to index a tensor: $A_{\mathsf{foo}(1)}$.
  - Use `\nmov{foo}{bar}{A}` for renaming: $[A]_{\mathsf{foo} \to \mathsf{bar}}$.
- Binary operators
  - Use `A \ndot{foo} B` for contraction: $A \underset{\mathsf{foo}}{\odot} B$.
  - Use `A \ncat{foo} B` for concatenation: $A \underset{\mathsf{foo}}{\oplus} B$.
  - In general, you can use `\nbin` to make a new binary operator with a name under it: `A \nbin{foo}{\star} B` gives you $A \underset{\mathsf{foo}}{\star} B$.
- Functions
  - Use `\nsum{foo} A` for summation: $\sum_{\mathsf{foo}} A$.
  - In general, you can use `\nfun` to make a function with a name under it: `\nfun{foo}{qux} A` gives you $\underset{\mathsf{foo}}{\mathrm{qux}}\, A$.

5 Formal Definitions

5.1 Records and shapes

A *named index* is a pair, written $\mathsf{ax}(i)$, where ax is a *name* and i is usually a natural number. We write both names and variables ranging over names using sans-serif font.

A *record* is a set of named indices $\{\mathsf{ax}_1(i_1), \ldots, \mathsf{ax}_r(i_r)\}$, where $\mathsf{ax}_1, \ldots \mathsf{ax}_r$ are pairwise distinct names.

An *axis* is a pair, written $\mathsf{ax}[I]$, where ax is a name and I is a set of *indices*.

We deal with axes of the form $\mathsf{ax}[[n]]$ (that is, $\mathsf{ax}[\{1, \ldots, n\}]$) so frequently that we abbreviate this as $\mathsf{ax}[n]$.

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 $|\operatorname{ind}(\mathsf{ax})|$.

A *shape* is a set of axes, written $\mathsf{ax}_1[I_1] \times \cdots \times \mathsf{ax}_r[I_r]$, where $\mathsf{ax}_1, \ldots \mathsf{ax}_r$ are pairwise distinct names. A shape defines a set of records:

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

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 $t|_{\mathcal{S}} \subseteq t$.

5.2 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 $\mathcal{S}$ 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 $A_{\{\mathsf{ax}_1(x_1),\ldots,\mathsf{ax}_r(x_r)\}}$, we usually write $A_{\mathsf{ax}_1(x_1),\ldots,\mathsf{ax}_r(x_r)}$. When a named tensor is an expression like $(A + B)$, we surround it with square brackets like this: $[A + B]_{\mathsf{ax}_1(x_1),\ldots,\mathsf{ax}_r(x_r)}$.

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.)

5.3 Extending functions to named tensors

In §2, we described several classes of functions that can be extended to named tensors. Here, we define how to do this for general functions.

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:

$$\begin{aligned} f &: F^{\mathcal{S}\cup\mathcal{U}} \to G^{\mathcal{T}\cup\mathcal{U}} \\ [f(A)]_u &= f(A_u) \qquad \text{for all } u \in \operatorname{rec}\mathcal{U}. \end{aligned}$$

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:

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

All of the tensor operations described in §2.2 can be defined in this way. For example, the contraction operator extends the following "named dot-product":

$$\begin{aligned} \underset{\mathsf{ax}}{\odot} &: F^{\mathsf{ax}[n]} \times F^{\mathsf{ax}[n]} \to F \\ A \underset{\mathsf{ax}}{\odot} B &= \sum_{i=1}^{n} A_{\mathsf{ax}(i)} B_{\mathsf{ax}(i)} \\ \underset{\mathsf{ax1}|\mathsf{ax2}}{\odot} &: F^{\mathsf{ax1}[n]} \times F^{\mathsf{ax2}[n]} \to F \\ A \underset{\mathsf{ax1}|\mathsf{ax2}}{\odot} B &= \sum_{i=1}^{n} A_{\mathsf{ax1}(i)} B_{\mathsf{ax2}(i)}. \end{aligned}$$

6 Extensions

6.1 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.

6.1.1 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:

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

6.1.2 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.

6.1.3 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, the RNN would look like this:

$$\begin{aligned}
x^t &\in \mathbb{R}^{\mathsf{input}} && t = 1, \ldots, n \\
W^{\mathrm{h}} &\in \mathbb{R}^{\mathsf{hidden}[d] \times \mathsf{hidden}[d]} \\
W^{\mathrm{i}} &\in \mathbb{R}^{\mathsf{input} \times \mathsf{hidden}[d]} \\
b &\in \mathbb{R}^{\mathsf{hidden}[d]} \\
h^0 &\in \mathbb{R}^{\mathsf{hidden}[d]} \\
h^t &= \sigma\left(W^{\mathrm{h}} \underset{\mathsf{hidden}}{\circ} h^{t-1} + W^{\mathrm{i}} \underset{\mathsf{input}}{\odot} x^t + b\right) && t = 1, \ldots, n
\end{aligned}$$

where $\circ$ does matrix-matrix and matrix-vector multiplication.

This wouldn't really help with self-attention. The multivariate normal distribution would need a transpose.

6.2 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

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

and extending it to higher-order tensors in its second argument according to the rules in §5.3. 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 $\mathcal{S}$:

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

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:

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

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

Thanks to Ekin Akyürek, Colin McDonald, Adam Poliak, Matt Post, Chungchieh Shan, Nishant Sinha, and Yee Whye Teh for their input to this document (or the ideas in it).

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.