Named Tensor Notation

David Chiang and Sasha Rush

December 11, 2020

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 indices by *position*, making code bug-prone. Quite a few libraries have been developed to identify indices 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). (Some of these libraries also add types to indices, but here we are only interested in adding names.)

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 indices by name instead of by position. This document describes our proposal to do both.

As a preview, the above equation becomes

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

making it unambiguous which index 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 indices.

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 *named shape*, which looks like this:

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

which stands for records where index foo ranges from 1 to 2, and index bar ranges from 1 to 3. Again, the order of the elements, called *named index sets*, 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 named shape to the real numbers. For example, here is a tensor with shape $\mathsf{foo}[2] \times \mathsf{bar}[3]$.

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

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

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

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 $\mathbb{R}^{m \times n}$ (which makes sense because one sometimes writes Y^X for the set of all functions from X to Y), we write $\mathbb{R}^{\mathsf{foo}[2] \times \mathsf{bar}[3]}$ for the set of all tensors with shape $\mathsf{foo}[2] \times \mathsf{bar}[3]$.

It's very common for an index name to be used with only one size. If you write

$$\mathsf{foo} \overset{\text{ind}}{=} [2]$$
$$\mathsf{bar} \overset{\text{ind}}{=} [3]$$

then you can simply write $\mathbb{R}^{\mathsf{foo} \times \mathsf{bar}}$ for the set of tensors with index foo ranging from 1 to 2 and bar ranging from 1 to 3. For more information on what $\overset{\text{ind}}{=}$ does, please see §5.

What are good choices for index 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 indices, 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:

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

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

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

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 index with the same name (and size), the two indices are *aligned*, as above. But if A has an index named i 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 index i.

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

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

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

Similarly, if B has an index named i and A doesn't, then we effectively replace A with a new tensor A' that contains a copy of A for every value of index i. 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 index 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,

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

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

More reductions: If A has shape $\mathsf{foo}[I] \times \ldots$, then

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

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

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

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

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

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

(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 indices at once.

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, which performs elementwise multiplication, then sums along an index. It can

be used, for example, for matrix multiplication:

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

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

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.

2.2.4 Vectors to vectors

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

$$\underset{\mathsf{foo}}{\operatorname{softmax}} A = \frac{\exp A}{\sum\limits_{\mathsf{foo}} \exp A} \approx \begin{matrix} \\ \mathsf{foo} \end{matrix} \begin{matrix} \mathsf{bar} \\ \begin{bmatrix} 0.731 & 0.002 & 0.953 \\ 0.269 & 0.998 & 0.047 \end{bmatrix} \end{matrix}.$$

And it's also very handy to have a function that renames an index:

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

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 Matrices

Finally, we briefly consider functions on matrices, for which you have to give *two* index 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 $\operatorname{inv}_{\mathsf{foo},\mathsf{bar}}$.

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

We use rectified linear units (ReLUs) because some of the larger models we define later require it; a logistic function or tanh would also work here.

$$\text{Full}\colon \mathbb{R}^{\mathsf{layer}} \to \mathbb{R}^{\mathsf{layer}'}$$

$$\text{Full}(x; W, b) = \max\left\{0, W \underset{\mathsf{layer}}{\cdot} x + b\right\}$$

where

$$W \in \mathbb{R}^{\mathsf{layer}' \times \mathsf{layer}}$$

$$b \in \mathbb{R}^{\mathsf{layer}'}$$

3.1.2 Recurrent neural networks

As a second example, let's define a simple (Elman) RNN. Let d be a positive integer.

$$\begin{aligned}
\mathsf{state}, \mathsf{state}' &\overset{\text{ind}}{=} [d] \\
x^{(t)} &\in \mathbb{R}^{\mathsf{emb}} && t = 1, \dots, n \\
h^{(t)} &\in \mathbb{R}^{\mathsf{state}} && t = 0, \dots, n \\
A &\in \mathbb{R}^{\mathsf{state} \times \mathsf{state}'} \\
B &\in \mathbb{R}^{\mathsf{emb} \times \mathsf{state}'} \\
c &\in \mathbb{R}^{\mathsf{state}'} \\
h^{(t+1)} &= \left[\tanh \left(A \underset{\mathsf{state}}{\cdot} h^{(t)} + B \underset{\mathsf{emb}}{\cdot} x^{(t)} + c \right) \right]_{\mathsf{state}' \to \mathsf{state}}
\end{aligned}$$

Here the index 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 (A) to one index and put the result in the same index. For possible solutions, see §6.2.

3.1.3 Attention

Let d_k and d_v be positive integers, and let n be the input sequence length.

$$\begin{aligned}
\mathsf{key} &\overset{\text{ind}}{=} [d_k] \\
\mathsf{val} &\overset{\text{ind}}{=} [d_v] \\
\mathrm{Att}_n &\colon \mathbb{R}^{\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}[n] \times \mathsf{key}} \times \mathbb{R}^{\mathsf{seq}[n] \times \mathsf{val}} \to \mathbb{R}^{\mathsf{val}} \\
\mathrm{Att}_n(Q, K, V) &= \underset{\mathsf{seq}}{\mathrm{softmax}} \left(\frac{Q \underset{\mathsf{key}}{\cdot} K}{\sqrt{d_k}} \right) \underset{\mathsf{seq}}{\cdot} V
\end{aligned}$$

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

$$\begin{aligned}
\mathrm{Att}_n &\colon \mathbb{R}^{\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}[n] \times \mathsf{key}} \times \mathbb{R}^{\mathsf{seq}[n] \times \mathsf{val}} \times \mathbb{R}^{\mathsf{seq}} \to \mathbb{R}^{\mathsf{val}} \\
\mathrm{Att}_n(Q, K, V, M) &= \underset{\mathsf{seq}}{\mathrm{softmax}} \left(\frac{Q \underset{\mathsf{key}}{\cdot} K}{\sqrt{d_k}} + M \right) \underset{\mathsf{seq}}{\cdot} V
\end{aligned}$$

If Q, K, and V have a head index for multiple attention heads, then the above equations will compute multi-head attention without modification.

3.1.4 Convolution

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

$$\begin{aligned}
\mathsf{kernel} &\overset{\text{ind}}{=} [k] \\
X &\in \mathbb{R}^{\mathsf{channels} \times \mathsf{seq}[n]} \\
W &\in \mathbb{R}^{\mathsf{channels} \times \mathsf{kernel}} \qquad\qquad b \in \mathbb{R} \\
\text{conv1d}(X; W, b) &= W \underset{\mathsf{channels},\mathsf{kernel}}{\cdot} U + b \\
U &\in \mathbb{R}^{\mathsf{channels} \times \mathsf{seq}[n-k+1] \times \mathsf{kernel}} \\
U_{\mathsf{seq}(i),\mathsf{kernel}(j)} &= X_{\mathsf{seq}(i+j-1)}
\end{aligned}$$

A 2-dimensional convolution:

$$\begin{aligned}
\mathsf{kh} &\overset{\text{ind}}{=} [kh] \\
\mathsf{kw} &\overset{\text{ind}}{=} [kw] \\
X &\in \mathbb{R}^{\mathsf{channels} \times \mathsf{height}[h] \times \mathsf{width}[w]} \\
W &\in \mathbb{R}^{\mathsf{channels} \times \mathsf{kh} \times \mathsf{kw}} \qquad\qquad b \in \mathbb{R} \\
\text{conv2d}(X; W, b) &= W \underset{\mathsf{channels},\mathsf{kh},\mathsf{kw}}{\cdot} U + b \\
U &\in \mathbb{R}^{\mathsf{channels} \times \mathsf{height}[h-kh+1] \times \mathsf{width}[w-kw+1] \times \mathsf{kh} \times \mathsf{kw}} \\
U_{\mathsf{height}(i),\mathsf{width}(j),\mathsf{kh}(ki),\mathsf{kw}(kj)} &= X_{\mathsf{height}(i+ki-1),\mathsf{width}(j+kj-1)}
\end{aligned}$$

3.1.5 Max pooling

$$\begin{aligned}
X &\in \mathbb{R}^{\mathsf{height}[h] \times \mathsf{width}[w]} \\
\text{maxpool2d}(X, kh, kw) &= \max_{\mathsf{kh},\mathsf{kw}} U \\
U &\in \mathbb{R}^{\mathsf{height}[h/kh] \times \mathsf{width}[w/kw] \times \mathsf{kh}[kh] \times \mathsf{kw}[kw]} \\
U_{\mathsf{height}(i),\mathsf{width}(j),\mathsf{kh}(di),\mathsf{kw}(dj)} &= X_{\mathsf{height}(i \times kh + di - 1),\mathsf{width}(j \times kw + dj - 1)}
\end{aligned}$$

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, parameterized by a named index set d:

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

where

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

Now, suppose that the input has three indices:

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

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

$Y = \underset{\mathsf{batch}}{\mathrm{xnorm}}(X)$	batch normalization
$Y = \underset{\mathsf{layer}}{\mathrm{xnorm}}(X)$	instance normalization
$Y = \underset{\mathsf{layer},\mathsf{channels}}{\mathrm{xnorm}}(X)$	layer normalization

3.2 Discrete random variables

Named indices are very helpful for working with discrete random variables, because each random variable can be represented by an index 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:

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

Then Bayes' rule is just:

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

3.3 Transformer

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

and positional encodings. Let d_{model} be a positive integer.

$$\begin{aligned}
\mathsf{layer} &\overset{\text{ind}}{=} [d_{\text{model}}] \\
I &\in \{0,1\}^{\mathsf{seq}[n] \times \mathsf{vocab}} && \sum_{\mathsf{vocab}} I = 1 \\
E &\in \mathbb{R}^{\mathsf{vocab} \times \mathsf{layer}} \\
W &= (E \underset{\mathsf{vocab}}{\cdot} I)\sqrt{d_{\text{model}}} \\
P &\in \mathbb{R}^{\mathsf{seq}[n] \times \mathsf{layer}} \\
P_{\mathsf{seq}(p),\mathsf{layer}(i)} &= \begin{cases} \sin((p-1)/10000^{(i-1)/d_{\text{model}}}) & i \text{ odd} \\ \cos((p-1)/10000^{(i-2)/d_{\text{model}}}) & i \text{ even} \end{cases}
\end{aligned}$$

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

$$\begin{aligned}
X^0 &= W + P \\
T^1 &= \underset{\mathsf{layer}}{\text{xnorm}}(\text{SelfAtt}(X^0)) + X^0 \\
X^1 &= \underset{\mathsf{layer}}{\text{xnorm}}(\text{FFN}(T^1)) + T^1 \\
&\vdots \\
T^L &= \underset{\mathsf{layer}}{\text{xnorm}}(\text{SelfAtt}(X^{L-1})) + X^{L-1} \\
X^L &= \underset{\mathsf{layer}}{\text{xnorm}}(\text{FFN}(T^L)) + T^L \\
O &= \underset{\mathsf{vocab}}{\text{softmax}}(E \underset{\mathsf{layer}}{\cdot} X^L)
\end{aligned}$$

where SelfAtt and FFNN are defined below.

The feedforward neural networks are:

$$\begin{aligned}
W^1 &\in \mathbb{R}^{\mathsf{hidden},\mathsf{layer}} & b^1 &\in \mathbb{R}^{\mathsf{hidden}} \\
W^2 &\in \mathbb{R}^{\mathsf{layer},\mathsf{hidden}} & b^2 &\in \mathbb{R}^{\mathsf{layer}} \\
\text{FFN}&\colon \mathbb{R}^{\mathsf{layer}} \to \mathbb{R}^{\mathsf{layer}} \\
\text{FFN}(x; W^1, b^1, W^2, b^2) &= \text{ReLU}(W^2 \underset{\mathsf{hidden}}{\cdot} y + b^2) \\
y &= \text{ReLU}(W^1 \underset{\mathsf{layer}}{\cdot} x + b^1)
\end{aligned}$$

We defined attention above (§3.1.3); the Transformer uses multi-head self-attention, in which Q, K, and V are all computed from the same sequence.

The parameters are:

$$\begin{aligned}
\mathsf{heads} &\overset{\text{ind}}{=} [h] \\
\mathsf{key}, \mathsf{val} &\overset{\text{ind}}{=} [d_{\text{model}}/h] \\
W^Q &\in \mathbb{R}^{\mathsf{heads} \times \mathsf{layer} \times \mathsf{key}} \\
W^K &\in \mathbb{R}^{\mathsf{heads} \times \mathsf{layer} \times \mathsf{key}} \\
W^V &\in \mathbb{R}^{\mathsf{heads} \times \mathsf{layer} \times \mathsf{val}} \\
W^O &\in \mathbb{R}^{\mathsf{heads} \times \mathsf{val} \times \mathsf{layer}}
\end{aligned}$$

Then define

$$\begin{aligned}
\text{SelfAtt}_n &\colon \mathbb{R}^{\mathsf{seq}[n] \times \mathsf{layer}} \to \mathbb{R}^{\mathsf{seq}[n] \times \mathsf{layer}} \\
\text{SelfAtt}_n(X; W^Q, W^K, W^V, W^O) &= \sum_{\mathsf{heads}} W^O \underset{\mathsf{val}}{\cdot} [\text{Att}(Q, K, V, M)]_{\mathsf{seq}' \to \mathsf{seq}}
\end{aligned}$$

where

$$\begin{aligned}
Q &= \left[W^Q \underset{\mathsf{layer}}{\cdot} X \right]_{\mathsf{seq} \to \mathsf{seq}'} \\
K &= W^K \underset{\mathsf{layer}}{\cdot} X \\
V &= W^V \underset{\mathsf{layer}}{\cdot} X \\
M &\in \mathbb{R}^{\mathsf{seq}[n] \times \mathsf{seq}'[n]} \\
M_{\mathsf{seq}(i), \mathsf{seq}'(j)} &= \begin{cases} 0 & i \le j \\ -\infty & \text{otherwise.} \end{cases}
\end{aligned}$$

3.4 LeNet

Parameters:

$$\begin{aligned}
W^1 &\in \mathbb{R}^{\mathsf{channels}'[c_2] \times \mathsf{channels}[c_1] \times \mathsf{kh}[kh_1] \times \mathsf{kw}[kw_1]} & b^1 &\in \mathbb{R}^{\mathsf{channels}'[c_2]} \\
W^2 &\in \mathbb{R}^{\mathsf{channels}'[c_3] \times \mathsf{channels}[c_2] \times \mathsf{kh}[kh_2] \times \mathsf{kw}[kw_2]} & b^2 &\in \mathbb{R}^{\mathsf{channels}'[c_3]}
\end{aligned}$$

Model:

$$
\begin{aligned}
X^0 &\in \mathbb{R}^{\mathsf{batch} \times \mathsf{channels}[c_1] \times \mathsf{height} \times \mathsf{width}} \\
X^1 &= \left[\mathrm{ReLU}(\mathrm{conv2d}(X^0; W^1, b^1))\right]_{\mathsf{channels}' \to \mathsf{channels}} \\
X^2 &= \mathrm{maxpool2d}(X^1, ph_1, ph_2) \\
X^3 &= \left[\mathrm{ReLU}(\mathrm{conv2d}(X^2; W^2, b^2))\right]_{\mathsf{channels}' \to \mathsf{channels}} \\
X^4 &= \mathrm{maxpool2d}(X^3, ph_2, ph_2) \\
X^5 &= \left[X^4\right]_{\mathsf{height},\mathsf{width},\mathsf{channels} \to \mathsf{layer}} \\
H &= \mathrm{ReLU}(W^3 \underset{\mathsf{layer}}{\cdot} X^5 + b^3) \\
O &= \underset{\mathsf{class}}{\mathrm{softmax}}(\mathrm{ReLU}(W^4 \underset{\mathsf{hidden}}{\cdot} H + b^4))
\end{aligned}
$$

The flattening operation in the equation for X^5 is defined in §**??**.

3.5 Other examples

3.5.1 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.

$$
\begin{aligned}
\mathsf{vocab} &\overset{\mathrm{ind}}{=} [v] \\
\mathsf{hidden} &\overset{\mathrm{ind}}{=} [h] \\
\mathsf{classes} &\overset{\mathrm{ind}}{=} [c] \\
X &\in \{0,1\}^{\mathsf{seq}[n] \times \mathsf{vocab}} \qquad \sum_{\mathsf{vocab}} X = 1 \\
E &\in \mathbb{R}^{\mathsf{vocab} \times \mathsf{hidden}} \\
W &\in \mathbb{R}^{\mathsf{classes} \times \mathsf{hidden}} \\
\mathrm{cbow}(X; E, W) &= \underset{\mathsf{class}}{\mathrm{softmax}}(W \underset{\mathsf{hidden}}{\cdot} E \underset{\mathsf{vocab}}{\cdot} X)
\end{aligned}
$$

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

3.5.2 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

$$X \in \{0,1\}^{\mathsf{height}[9] \times \mathsf{width}[9] \times \mathsf{assign}[9]}$$

$$\text{check}(X) = \left(\sum_{\mathsf{assign}} Y = \sum_{\mathsf{Height},\mathsf{height}} Y = \sum_{\mathsf{Width},\mathsf{width}} Y = \sum_{\mathsf{height},\mathsf{width}} Y = 1 \right)$$

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

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

3.5.3 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,

$$\begin{aligned} \mathsf{batch} &\overset{\text{ind}}{=} [b] \\ \mathsf{space} &\overset{\text{ind}}{=} [k] \\ \mathsf{clusters} &\overset{\text{ind}}{=} [c] \\ X &\in \mathbb{R}^{\mathsf{batch} \times \mathsf{space}} \\ C &\in \mathbb{R}^{\mathsf{clusters} \times \mathsf{space}} \end{aligned}$$

we compute cluster assignments

$$\begin{aligned} Q &= \underset{\mathsf{clusters}}{\text{argmin}} \underset{\mathsf{space}}{\text{norm}}(C - X) \\ &= \lim_{\alpha \to -\infty} \underset{\mathsf{clusters}}{\text{softmax}} \left(\alpha \underset{\mathsf{space}}{\text{norm}}(C - X) \right) \end{aligned}$$

then we recompute the cluster centers:

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

3.5.4 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. Beam step returns the new H tensor.

$$
\begin{aligned}
\mathsf{batch} &\overset{\text{ind}}{=} [b] \\
\mathsf{beam} &\overset{\text{ind}}{=} [k] \\
\mathsf{state}, \mathsf{state}' &\overset{\text{ind}}{=} [s] \\
H &\in \mathbb{R}^{\mathsf{batch} \times \mathsf{beam}} \\
S &\in \{0,1\}^{\mathsf{batch} \times \mathsf{beam} \times \mathsf{state}} \qquad \sum_{\mathsf{state}} S = 1 \\
f &\colon \{0,1\}^{\mathsf{state}} \to \mathbb{R}^{\mathsf{state}'} \\
\text{beamstep}(H, S) &= \underset{\mathsf{beam},\mathsf{state}'}{\text{maxk}} \left(\underset{\mathsf{state}'}{\text{softmax}}(f(S)) \odot H \right)
\end{aligned}
$$

3.5.5 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).

$$
\begin{aligned}
\mathsf{batch} &\overset{\text{ind}}{=} [b] \\
\mathsf{d}, \mathsf{d1}, \mathsf{d2} &\overset{\text{ind}}{=} [k] \\
X &\in \mathbb{R}^{\mathsf{batch} \times \mathsf{d}} \\
\mu &\in \mathbb{R}^{\mathsf{d}} \\
\Sigma &\in \mathbb{R}^{\mathsf{d1} \times \mathsf{d2}} \\
\mathcal{N}(X; \mu, \Sigma) &= \frac{\exp\left(-\frac{1}{2} \left(\underset{\mathsf{d1},\mathsf{d2}}{\text{inv}}(\Sigma) \underset{\mathsf{d1}}{\cdot} [X-\mu]_{\mathsf{d} \to \mathsf{d1}} \right) \underset{\mathsf{d2}}{\cdot} [X-\mu]_{\mathsf{d} \to \mathsf{d2}} \right)}{\sqrt{(2\pi)^k \underset{\mathsf{d1},\mathsf{d2}}{\det}(\Sigma)}}
\end{aligned}
$$

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:

- Use `\name{foo}` to write an index name: foo.
- Use `\ndef` to define a named index set: $\mathsf{foo} \overset{\text{ind}}{=} [n]$.

- 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: $[A]_{\mathsf{foo}\to\mathsf{bar}}$.

5 Formal Definitions

5.1 Records

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

A *record* is a set of named indices $\{\mathsf{i}_1(x_1), \ldots, \mathsf{i}_r(x_r)\}$, where $\mathsf{i}_1, \ldots \mathsf{i}_r$ are pairwise distinct names.

If i is a name and X is a set, define the *named index set* $\mathsf{i}[X]$ to be the set $\{\mathsf{i}(x) \mid x \in X\}$. We deal with named index sets of the form $\mathsf{i}[[n]]$ $(= \mathsf{i}[\{1, \ldots, n\}])$ so frequently that we abbreviate this as $\mathsf{i}[n]$.

A *named shape* is a set of named index sets $\{\mathsf{i}_1[X_1], \ldots, \mathsf{i}_r[X_r]\}$, where $\mathsf{i}_1, \ldots \mathsf{i}_r$ are pairwise distinct names. A named shape defines a set of records:

$$\mathrm{ind}\{\mathsf{i}_1[X_1], \ldots, \mathsf{i}_r[X_r]\} = \{\{\mathsf{i}_1(x_1), \ldots, \mathsf{i}_r(x_r)\} \mid x_1 \in X_1, \ldots, x_r \in X_r\}\,,$$

and for this reason we often write a shape with the alternative notation $\mathsf{i}_1[X_1] \times \cdots \times \mathsf{i}_r[X_r]$, because it's like an "unordered Cartesian product" of the index sets.

We say two shapes $\mathcal{S}$ and $\mathcal{T}$ are *compatible* if whenever $\mathsf{i}(X) \in \mathcal{S}$ and $\mathsf{i}(Y) \in \mathcal{T}$, then $X = Y$. We say that $\mathcal{S}$ and $\mathcal{T}$ are *orthogonal* if there is no i such that $\mathsf{i}(X) \in \mathcal{S}$ and $\mathsf{i}(Y) \in \mathcal{T}$ for any X, Y.

If $t \in \mathrm{ind}\,\mathcal{T}$ and $\mathcal{S} \subseteq \mathcal{T}$, then we write $t|_{\mathcal{S}}$ for the unique tuple in $\mathrm{ind}\,\mathcal{S}$ such that $t|_{\mathcal{S}} \subseteq t$.

Because it's very common for a name to be always used with the same index set, we can write $\mathsf{i} \overset{\mathrm{ind}}{=} X$ to create a named index set $\mathsf{i}[X]$, and this named index set is *also* called i. The context always makes it clear whether the name or the named index set is meant.

5.2 Named tensors

Let F be a field and let $\mathcal{T}$ be a set of records. Then a *named tensor over F with shape $\mathcal{T}$* is a mapping from $\mathcal{T}$ to F. We write the set of all named tensors

with shape $\mathcal{T}$ as $F^{\mathcal{T}}$.

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 $A \in F^{\mathcal{T}}$, then we access an element of A by applying it to a record $t \in \operatorname{ind}\mathcal{T}$; but we write this using the usual subscript notation: A_t rather than $A(t)$. To avoid clutter, in place of $A_{\{\mathsf{i}_1(x_1),\ldots,\mathsf{i}_r(x_r)\}}$, we usually write $A_{\mathsf{i}_1(x_1),\ldots,\mathsf{i}_r(x_r)}$. When a named tensor is an expression like $(A+B)$, we surround it with square brackets like this: $[A+B]_{\mathsf{i}_1(x_1),\ldots,\mathsf{i}_r(x_r)}$.

We also allow partial indices. If A is a tensor with shape $\mathcal{T}$ and $s \in \operatorname{ind}\mathcal{S}$ 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 $t \in \operatorname{ind}(\mathcal{T} \setminus \mathcal{S})$,

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

(For the edge case $\mathcal{S} = \mathcal{U}$ and $\mathcal{T} = \emptyset$, 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:

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

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:

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

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":

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

6 Extensions

6.1 Index types

We have defined a named index set as a pair $\mathsf{i}[X]$, where i is a name and X is a set, usually $[n]$ for some n. In this section, we consider some other possibilities for X.

6.1.1 Non-integral types

The sets X 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 index from being aligned with another index 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, instead of defining an inv operator that takes two subscripts, we could write

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

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

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

6.2 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 §3.1.2 we saw that defining an RNN requires renaming of indices, because a linear transformation must map one index to another index; if we want to map an index 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.

6.2.1 Contracting two names

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

$$\underset{\mathsf{i}|\mathsf{j}}{\cdot} : F^{\mathsf{i}[n]} \times F^{\mathsf{j}[n]} \to F$$

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

For example, the RNN would look like this.

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

6.2.2 Starred index names

If i is a name, we also allow a tensor to have an index $\mathsf{i}*$ (alternatively: superscript i). Multiplication contracts starred indices in the left operand with

non-starred indices in the right operand.

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

The contraction operator can be defined as:

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

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

1. $\cdot$ (no subscript) contracts every starred index in its left operand with every corresponding unstarred index in its right operand. Rejected.
2. $\underset{\mathsf{i}}{\cdot}$ contracts i with i, and we need another notation like $\underset{\mathsf{i}(*)}{\cdot}$ or $\underset{\mathsf{i}}{\times}$ for contracting $\mathsf{i}*$ with i.
3. $\underset{\mathsf{i}}{\cdot}$ always contracts $\mathsf{i}*$ with i; there's no way to contract i with i.

6.2.3 Named and numbered indices

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

A_{i}	rename index i to 1
$A_{\mathsf{i},\mathsf{j}}$	rename index i to 1 and j to 2
$A_{\to\mathsf{i}}$	rename index 1 to i
$A_{\to\mathsf{i},\mathsf{j}}$	rename index 1 to i and 2 to j

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

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

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

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

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 $\max_{\mathsf{i}}$), but occasionally it looked quite different (e.g., matrix inversion). With numbered indices, 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: $A_{\mathsf{i}\to\mathsf{j}} = [A_{\mathsf{i}}]_{\to\mathsf{j}}$ renames index i to j.

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:

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

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

Multivariate normal distribution:

$$\begin{aligned} X &\in \mathbb{R}^{\mathsf{batch}[b]\times\mathsf{d}[k]} \\ \mu &\in \mathbb{R}^{\mathsf{d}[k]} \\ \Sigma &\in \mathbb{R}^{\mathsf{d}[k]\times\mathsf{d}'[k]} \\ \mathcal{N}(X;\mu,\Sigma) &= \frac{\exp\left(-\frac{1}{2}[X-\mu]^{\top}_{\mathsf{d}}\, \Sigma^{-1}_{\mathsf{d},\mathsf{d}'}\, [X-\mu]_{\mathsf{d}}\right)}{\sqrt{(2\pi)^k\, \det \Sigma_{\mathsf{d},\mathsf{d}'}}} \end{aligned}$$

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

6.3 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 $I^1, \dots, I^D$ are named tensors, called "indexers," what is $A_{I^1,\dots,I^D}$?

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

$$\begin{aligned} &A \in F^{\mathsf{i}_1[X_1],\dots,\mathsf{i}_D[X_D]} \\ &I^d \in X_D^{\mathcal{S}} && d = 1, \dots, D. \end{aligned}$$

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

$$[A_{I^1,\dots,I^D}]_s = A_{I^1_s,\dots,I^D_s}.$$

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

$$\begin{aligned} &A \in F^{\mathsf{i}_1[X_1],\dots,\mathsf{i}_D[X_D]} \\ &I^d \in X_D^{\mathcal{S}_d} && d = 1, \dots, D, \end{aligned}$$

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

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

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 $X \in \mathbb{R}^{\mathsf{batch}[B],\mathsf{sent}[N],\mathsf{emb}[E]}$. 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 $Y \in \mathbb{R}^{\mathsf{batch}[B],\mathsf{span}[M],\mathsf{emb}[E]}$.

We create a indexer for the sent axis: $I_{\mathsf{sent}} \in [N]^{\mathsf{batch}:B \times \mathsf{span}:M}$ that selects the desired tokens. Then we can write

$$Y = X_{\mathsf{batch}[I],\mathsf{sent}[I],\mathsf{emb}[I]}$$

where the other two indexers

$$\begin{aligned} I_{\mathsf{batch}} &\in [B]^{\mathsf{batch}[B]} \\ [I_{\mathsf{batch}}]_{\mathsf{batch}[b]} &= b \\ I_{\mathsf{emb}} &\in [E]^{\mathsf{emb}[E]} \\ [I_{\mathsf{sent}}]_{\mathsf{sent}[n]} &= n \end{aligned}$$

select all values of their respective indices.

Acknowledgements

Thanks to Ekin Akyürek, Colin McDonald, Chung-chieh Shan, and Nishant Sinha 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, St'efan 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 Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-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.