Named Tensor Notation

David Chiang
University of Notre Dame

Sasha Rush
Cornell University

Boaz Barak
Harvard University

Version 0.4

Abstract

We propose a notation for tensors with named axes, which relieves the author, reader, and future implementers from the burden of keeping track of the order of axes and the purpose of each. It also makes it easy to extend operations on low-order tensors to higher order ones (e.g., to extend an operation on images to minibatches of images, or extend the attention mechanism to multiple attention heads).

After a brief overview of our notation, we illustrate it through several examples from modern machine learning, from building blocks like attention and convolution to full models like Transformers and LeNet. Finally, we give formal definitions and describe some extensions. Our proposals build on ideas from many previous papers and software libraries. We hope that this document will encourage more authors to use named tensors, resulting in clearer papers and less bug-prone implementations.

The source code for this document can be found at `https://github.com/namedtensor/notation/`. We invite anyone to make comments on this proposal by submitting issues or pull requests on this repository.

Contents

1 Introduction

Most papers about neural networks use the notation of vectors and matrices from applied linear algebra. This notation is optimized for talking about vector spaces, but becomes cumbersome when talking about neural networks. Consider the following equation (Vaswani et al., 2017):

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

where Q, K, and V (for query, key, and value, respectively) are sequences of feature vectors, packed into matrices. Does the product $QK^\top$ sum over the sequence, or over the features? It sums over columns, but there's not enough information to know what the columns represent. 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 or multiple sentences in a minibatch, the notation becomes more difficult still.

Here, we propose mathematical notation for tensors with *named axes*. The notation has a formal underpinning, but is hopefully intuitive enough that machine learning researchers can understand it without much effort.

In our notation, the above equation becomes

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

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

The tensor K has axes for the sequence (seq) and for the key features (key), instead of rows or columns, so the reader does not need to remember which is which. The dot product $Q \underset{\mathsf{key}}{\odot} K$ is explicitly over the key axis. The resulting tensor has a seq axis for the key sequence and a seq' axis for the query sequence, and the softmax is explicitly over seq, as is the dot product with V. This formula works as written if we add a heads axis for multiple attention heads, or a batch axis for multiple sequences in a minibatch.

Our notation is inspired by libraries for programming with multidimensional arrays (Harris et al., 2020; Paszke et al., 2019) and extensions that use named axes, like xarray (Hoyer and Hamman, 2017), Nexus (Chen, 2017), tsalib (Sinha, 2018), NamedTensor (Rush, 2019), named tensors in PyTorch (Torch Contributors, 2019), and Dex (Maclaurin et al., 2019). However, our focus is on mathematical notation rather than code.

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

In standard notation, a vector, matrix, or tensor is indexed by an integer or sequence of integers. If $A \in \mathbb{R}^{3\times 3}$, then the order of the two axes matters: $A_{1,3}$ and $A_{3,1}$ are not the same element. It's up to the reader to remember what each axis of each tensor is for. We think this is a problem and propose a solution.

2.1 Named tensors

In a *named tensor*, we give each axis a name. For example, if A represents an image, we can make it a named tensor like so (writing it two equivalent ways to show that the order of axes does not matter):

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

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

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

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

In many contexts, an axis name is used with only one size. If so, we can simply write height for the unique axis with name height, as in $\mathbb{R}^{\mathsf{height}\times\mathsf{width}}$. We can leave the size of an axis unspecified at first, and specify its size later (like in a section on experimental details): for example, $|\mathsf{height}| = |\mathsf{width}| = 28$ to specify its exact size or just $|\mathsf{height}| = |\mathsf{width}|$ to specify that it's a square image.

What are good choices for axis names? We recommend meaningful *words* instead of single letters, and we recommend words that describe a *whole* rather than its parts. For example, if we wanted A to have red, green, and blue channels, we'd name the axis chans, and if we wanted to represent a minibatch of images, we'd name the axis batch. Please see §3 for more examples.

2.2 Named tensor operations

Operations on named tensors are defined by taking a function on low-order tensors and extending it to higher-order tensors.

2.2.1 Elementwise operations and broadcasting

Any function from a scalar to a scalar can be applied elementwise to a named tensor, and any function from two scalars to a scalar can be applied to two

named tensors with the same shape. For example:

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

But if we apply a binary function/operator to tensors with different shapes, they are *broadcast* against each other (similarly to NumPy and derivatives). Let

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

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

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

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

2.2.2 Reductions

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

See §5.4 for some common reductions. Here we take summation as an example.

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

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

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

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

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

The vector dot-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 it as elementwise multiplication, then summation over one axis:

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

Again, we can write multiple names to contract multiple axes at once. A $\odot$ with no axis name under it contracts zero axes and is equivalent to elementwise multiplication, so we use $\odot$ for elementwise multiplication as well.

The contraction operator can be used for many multiplication-like operations:

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

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

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

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

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

2.2.3 Renaming and reshaping

It's often useful to rename an axis (analogous to a transpose operation in standard notation):

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

We can also reshape two or more axes into one axis:

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

The order of elements in the new axis is undefined. If you need a particular order, you can write a more specific definition.

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. Many of these examples use functions that the reader can probably guess the meaning of; if not, please see §5.4 for definitions.

3.1 Building blocks

3.1.1 Feedforward neural networks

A feedforward neural network looks like this:

$$
\begin{aligned}
&X^0 \in \mathbb{R}^{\mathsf{input}} \\
&X^1 = \sigma(W^1 \underset{\mathsf{input}}{\odot} X^0 + b^1) && W^1 \in \mathbb{R}^{\mathsf{hidden}_1 \times \mathsf{input}} && b^1 \in \mathbb{R}^{\mathsf{hidden}_1} \\
&X^2 = \sigma(W^2 \underset{\mathsf{hidden}_1}{\odot} X^1 + b^2) && W^2 \in \mathbb{R}^{\mathsf{hidden}_2 \times \mathsf{hidden}_1} && b^2 \in \mathbb{R}^{\mathsf{hidden}_2} \\
&X^3 = \sigma(W^3 \underset{\mathsf{hidden}_2}{\odot} X^2 + b^3) && W^3 \in \mathbb{R}^{\mathsf{out} \times \mathsf{hidden}_2} && b^3 \in \mathbb{R}^{\mathsf{out}}
\end{aligned}
$$

The layer sizes can be set by writing $|\mathsf{input}| = 100$, etc.

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

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

where

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

A couple of things are new here. First, $\mathrm{FullConn}^l$ encapsulates both the equation for layer l as well as its parameters (analogous to what TensorFlow and PyTorch call *modules*). Second, we chose to use the same axis name layer for all the layers (with different sizes n_l). So $\mathrm{FullConn}^l$ temporarily computes its output over axis layer', then renames it back to layer.

Then the network can be defined like this:

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

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.

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

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:

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

This equation is slightly different from the one in the introduction. The previous definition computed an output sequence over axis seq', but this definition computes a single value. If we want a sequence, we can just give Q a seq' axis (or some other name), and the function will compute an output sequence. Furthermore, if we give Q, K, and V a heads axis for multiple attention heads, then the function will compute multi-head attention.

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

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

3.1.4 Convolution

Convolutions can be easily written by unrolling a tensor and then applying a standard dot product. First, we define the unrolling operation:

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

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

Then we can define convolutions as:

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

where

$$W \in \mathbb{R}^{\mathsf{chans}\times\mathsf{kernel}}$$
$$b \in \mathbb{R}$$

This computes a single output channel, but we can get multiple output channels by giving W and b a chans' axis (or some other name).

The same unrolling operation can be used to define higher-dimensional convolutions:

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

where

$$W \in \mathbb{R}^{\mathsf{chans}\times\mathsf{kh}\times\mathsf{kw}}$$
$$b \in \mathbb{R}.$$

3.1.5 Max pooling

We first define an operation to reshape an axis:

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

Then we can define:

$$\mathrm{MaxPool1d}_k\colon \mathbb{R}^{\mathsf{seq}[n]} \to \mathbb{R}^{\mathsf{seq}[n/k]}$$
$$\mathrm{MaxPool1d}_k(X) = \max_{\mathsf{kernel}}\, \underset{\mathsf{seq,kernel}}{\mathrm{pool}}\, X$$

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

Other pooling functions could be defined similarly.

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 both by which axes are *standardized* as well as their parameters.

We can define a single generic standardization function as:

$$
\begin{aligned}
\underset{\mathsf{ax}}{\text{standardize}} &\colon \mathbb{R}^{\mathsf{ax}} \to \mathbb{R}^{\mathsf{ax}} \\
\underset{\mathsf{ax}}{\text{standardize}}(X) &= \frac{X - \underset{\mathsf{ax}}{\text{mean}}(X)}{\sqrt{\underset{\mathsf{ax}}{\text{var}}(X) + \epsilon}}
\end{aligned}
$$

where $\epsilon > 0$ is a small constant for numerical stability.

Then, we can define the three kinds of normalization layers, all with type $\mathbb{R}^{\mathsf{batch} \times \mathsf{chans} \times \mathsf{layer}} \to \mathbb{R}^{\mathsf{batch} \times \mathsf{chans} \times \mathsf{layer}}$:

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

Note that, while superficially similar, these functions differ in their standardized axes and their parameter shape.

Other deep learning methods have been proposed which consider different shapes of standardization. For instance, group norm is a popular extension that first pools channels into k-size groups before standardizing.

$$
\text{GroupNorm}_k(X; \gamma, \beta) = \left[\underset{\mathsf{kernel},\mathsf{layer}}{\text{standardize}} \; \underset{\mathsf{chans},\mathsf{kernel}}{\text{pool}} \; X \right]_{(\mathsf{chans},\mathsf{kernel}) \to \mathsf{chans}} \odot \gamma + \beta
$$

where

$$|\mathsf{kernel}| = k$$
$$\gamma, \beta \in \mathbb{R}^{\mathsf{chans}}.$$

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

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

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.

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

where

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

Feedforward neural networks:

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

where

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

3.3 LeNet

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

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

$$X^2 = \text{MaxPool}^2(T^2)$$

$$X^3 = \text{relu}(W^3 \underset{\substack{\mathsf{height}\\ \mathsf{width}\\ \mathsf{chans}}}{\odot} X^2 + b^3) \qquad W^3 \in \mathbb{R}^{\mathsf{hidden} \times \mathsf{height} \times \mathsf{width} \times \mathsf{chans}}.$$

The convolution and pooling operations are defined as follows:

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

where

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

and

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

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 Advanced indexing

Contributors: Tongfei Chen and Chu-Cheng Lin

NumPy and its derivatives provide various ways to recombine elements of a tensor to form a new tensor: integer array indexing, and functions like `take`,

`index_select`, `gather`, and `batch_gather`. Using named tensors, we can write nearly all of these operations with a single function:

$$\underset{\mathsf{ax}}{\text{index}} \colon \mathbb{R}^{\mathsf{ax}[n]} \times [n] \to \mathbb{R}$$
$$\underset{\mathsf{ax}}{\text{index}}(A, i) = A_{\mathsf{ax}(i)}.$$

Suppose we have

$$E \in \mathbb{R}^{\mathsf{vocab}[n] \times \mathsf{emb}}$$
$$i \in [n]$$
$$I \in [n]^{\mathsf{seq}}$$
$$P \in \mathbb{R}^{\mathsf{seq} \times \mathsf{vocab}[n]}$$

Tensor E contains word embeddings for all the words in the vocabulary. Integer i is the numeric identifier of a word, while tensor I is a sequence of words. Tensor P contains a sequence of probability distributions over the vocabulary. Then:

- The expression $\underset{\mathsf{vocab}}{\text{index}}(E, i)$ broadcasts E's emb axis, giving the word embedding of word i. This is the same as partial indexing ($E_{\mathsf{vocab}(i)}$).
- The expression $\underset{\mathsf{vocab}}{\text{index}}(E, I)$ also broadcasts I's seq axis, giving a sequence of word embeddings. This is the same as integer array indexing.
- The expression $\underset{\mathsf{vocab}}{\text{index}}(P, I)$ aligns P's and I's seq axes, giving a sequence of probabilities. This is the same as `gather`.

In NumPy, indexing using two or more integer arrays requires a special definition with some surprising special cases. With named tensors, we simply apply the indexing function twice. For example, if we (for some reason) wanted to get probabilities of words at a subset of positions:

$$|\mathsf{seq}| = m$$
$$I_1 = [m]^{\mathsf{subseq}}$$
$$I_2 = [n]^{\mathsf{subseq}}$$
$$S = \underset{\mathsf{vocab}}{\text{index}}(\underset{\mathsf{seq}}{\text{index}}(P, I_1), I_2) \in \mathbb{R}^{\mathsf{subseq}}$$
$$S_{\mathsf{subseq}(i)} = P_{\mathsf{seq}(I_{\mathsf{subseq}(i)}), \mathsf{vocab}(I_{\mathsf{subseq}(i)})}.$$

3.4.3 Continuous bag of words

A continuous bag-of-words model classifies by summing up the embeddings of a sequence of words X (as one-hot vectors) and projecting them to the space of classes.

$$\text{CBOW}\colon \{0,1\}^{\mathsf{seq}\times\mathsf{vocab}} \to \mathbb{R}^{\mathsf{classes}}$$

$$\text{CBOW}(X; E, W) = \underset{\mathsf{classes}}{\text{softmax}}\left(\sum_{\mathsf{seq}} W \underset{\mathsf{emb}}{\odot} E \underset{\mathsf{vocab}}{\odot} X\right)$$

where

$$\sum_{\mathsf{vocab}} X_{\mathsf{seq}(i)} = 1 \qquad\qquad i = 1, \ldots, |\mathsf{seq}|$$

$$E \in \mathbb{R}^{\mathsf{vocab}\times\mathsf{emb}}$$

$$W \in \mathbb{R}^{\mathsf{classes}\times\mathsf{emb}}.$$

3.4.4 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.

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

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

where

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

$$Y_{\mathsf{height}'(h'),\mathsf{height}(h),\mathsf{width}'(w'),\mathsf{width}(w)} = X_{\mathsf{height}(3h'+h-1),\mathsf{width}(3w'+w-1)}.$$

3.4.5 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,

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

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

we repeat the following update: Compute cluster assignments

$$Q = \underset{\mathsf{clusters}}{\text{argmin}}\, \underset{\mathsf{d}}{\text{norm}}(C - X)$$

then recompute the cluster centers:

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

3.4.6 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.

3.4.7 Multivariate normal distribution

To define a multivariate normal distribution, we need some matrix operations. These have two axis names written under them, for rows and columns, respectively. Determinant and inverse have the following signatures:

$$\det_{\mathsf{ax}_1,\mathsf{ax}_2} : F^{\mathsf{ax}_1[n] \times \mathsf{ax}_2[n]} \to F$$
$$\operatorname{inv}_{\mathsf{ax}_1,\mathsf{ax}_2} : F^{\mathsf{ax}_1[n] \times \mathsf{ax}_2[n]} \to F^{\mathsf{ax}_1[n] \times \mathsf{ax}_2[n]}.$$

(We write inv instead of $\cdot^{-1}$ because there's no way to write axis names under the latter.)

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).

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

$$\mathcal{N}(X;\mu,\Sigma) = \frac{\exp\left(-\frac{1}{2}\left(\underset{\mathsf{d}_1,\mathsf{d}_2}{\operatorname{inv}}\ \Sigma\right) \underset{\mathsf{d}_1,\mathsf{d}_2}{\odot} \left([X-\mu]_{\mathsf{d}\to\mathsf{d}_1} \odot [X-\mu]_{\mathsf{d}\to\mathsf{d}_2}\right)\right)}{\sqrt{(2\pi)^{|\mathsf{d}|} \underset{\mathsf{d}_1,\mathsf{d}_2}{\det}\ \Sigma}}$$

where

$$|\mathsf{d}| = |\mathsf{d}_1| = |\mathsf{d}_2|$$
$$\mu \in \mathbb{R}^{\mathsf{d}}$$
$$\Sigma \in \mathbb{R}^{\mathsf{d}_1 \times \mathsf{d}_2}.$$

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}`).

We write axis names in sans-serif font. To make this easier, `\ndef{\ax}{ax}` defines a macro `\ax` that looks like this: ax.

- Binary operators
  - Use `A \ndot{\ax} B` for contraction: $A \underset{\mathsf{ax}}{\odot} B$. You can use `\\` to stack up several names.
  - In general, you can use `\nbin` to make a new binary operator with a name under it: `A \nbin{\ax}{\star} B` gives you $A \underset{\mathsf{ax}}{\star} B$.
- Functions
  - Use `\nsum{\ax} A` for summation: $\sum_{\mathsf{ax}} A$.
  - In general, you can use `\nfun` to make a function with a name under it: `\nfun{\ax}{qux} A` gives you $\underset{\mathsf{ax}}{\operatorname{qux}} A$.

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. We write $\emptyset$ for the empty shape. A shape defines a set of records:

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

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 $A_{\{\mathsf{ax}_1(i_1),\ldots,\mathsf{ax}_r(i_r)\}}$, we usually write $A_{\mathsf{ax}_1(i_1),\ldots,\mathsf{ax}_r(x_r)}$. When a named tensor is an expression like $(A + B)$, we surround it with square brackets like this: $[A + B]_{\mathsf{ax}_1(i_1),\ldots,\mathsf{ax}_r(x_r)}$.

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 Named tensor operations

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 $\mathcal{S}'$ orthogonal to both $\mathcal{S}$ and $\mathcal{T}$, we can extend f to:

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

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 $\mathcal{S}' \cup \mathcal{T}'$ is orthogonal to $\mathcal{U}$, we can extend f to:

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

5.4 Common operations

All the tensor operations described in §2.2 can be defined in this way, and others listed below.

Elementwise operations $(\mathbb{R} \to \mathbb{R})$

$$\begin{aligned} \sigma(x) &= \frac{1}{1 + \exp(-x)} \\ \operatorname{relu}(x) &= \max(0, x) \end{aligned}$$

Reductions $(\mathbb{R}^{\mathsf{ax}[n]} \to \mathbb{R})$

$$\begin{aligned} \sum_{\mathsf{ax}} A &= \sum_{i=1}^{n} A_{\mathsf{ax}(i)} \\ \min_{\mathsf{ax}} A &= \min\{A_{\mathsf{ax}(i)} \mid 1 \le i \le n\} \end{aligned}$$

$$\max_{\mathsf{ax}} A = \max\{A_{\mathsf{ax}(i)} \mid 1 \le i \le n\}$$

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

$$\operatorname*{mean}_{\mathsf{ax}} A = \frac{1}{n} \sum_{\mathsf{ax}} A$$

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

Contraction $(\mathbb{R}^{\mathsf{ax}[n]} \times \mathbb{R}^{\mathsf{ax}[n]} \to F)$

$$A \operatorname*{\odot}_{\mathsf{ax}} B = \sum_{i=1}^{n} A_{\mathsf{ax}(i)} B_{\mathsf{ax}(i)}$$

Vectors to vectors $(\mathbb{R}^{\mathsf{ax}[n]} \to \mathbb{R}^{\mathsf{ax}[n]})$

$$\operatorname*{softmax}_{\mathsf{ax}} A = \frac{\exp A}{\sum_{\mathsf{ax}} \exp A}$$

$$\operatorname*{argmax}_{\mathsf{ax}} A = \lim_{\alpha \to \infty} \operatorname*{softmax}_{\mathsf{ax}} \alpha A$$

$$\operatorname*{argmin}_{\mathsf{ax}} A = \lim_{\alpha \to -\infty} \operatorname*{softmax}_{\mathsf{ax}} \alpha A$$

Renaming $(\mathbb{R}^{\mathsf{ax}[n]} \to \mathbb{R}^{\mathsf{ax}'[n]})$

$$[A_{\mathsf{ax} \to \mathsf{ax}'}]_{\mathsf{ax}'(i)} = A_{\mathsf{ax}(i)}$$

6 Differentiation

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

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

With tensors, taking derivatives of higher-order tensors with respect to higher-order tensors is not difficult (Laue et al., 2018). With named tensors, we get the additional advantage of using names to distinguish input and output axes.

6.1 Definition

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

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

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

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

6.2 Rules

To compute derivatives, we use the method of differentials (Magnus and Neudecker, 1985). The differential of an expression U, written ∂U, is a tensor with the same shape as U, computed using rules like the following:

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

If we obtain an equation of the form

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

where $\mathcal{S}$ is orthogonal to $\mathcal{T}$ and "const" stands for terms not depending on ∂X, then we have

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

In order to get equations into the form (1), some tricks are useful. First, contractions can be easier to reason about if rewritten as sums of elementwise

products:

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

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

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

6.3 Example

Let's find the differential of the softmax operator.

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

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

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

To derive the rule for backpropagation, we assume a function $f\colon \mathbb{R}^{\mathsf{ax}} \to \mathbb{R}$ and differentiate $f(Y)$. Since f is scalar-valued, there is no name overlap, so no

renaming is needed.

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

6.4 Broadcasting

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

$$Y_r = f(X_r)$$

Finding the differential of Y is easy:

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

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

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

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

7 Alternatives

A very frequently asked question is why we haven't used index notation as used in physics, and the Einstein summation convention in particular. In this

notation, axes are ordered, and every equation is written in terms of tensor components. If an index appears on both sides of an equation, then the equation must hold for each value of the index, and if an index appears twice on one side and not on the other, there is an implicit summation over that index.

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

$$[\text{Attention}(Q, K, V)]_{i'k} = \underset{i}{\text{softmax}}\left(\frac{Q_{i'j}K_{ij}}{\sqrt{d_k}}\right) V_{ik}.$$

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

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

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

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

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

Finally, index notation requires us to write out all indices explicitly. So if we wanted to extend attention to multiple heads and minibatches, we would write:

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

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

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

Acknowledgements

Thanks to Ekin Akyürek, Justin Bayer, Colin McDonald, Adam Poliak, Matt Post, Chung-chieh 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.

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

Soeren Laue, Matthias Mitterreiter, and Joachim Giesen. 2018. Computing higher order derivatives of matrix and tensor expressions. In *Advances in Neural Information Processing Systems*, volume 31, pages 2750–2759. Curran Associates, Inc.

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*.

Jan R. Magnus and H. Neudecker. 1985. Matrix differential calculus with applications to simple, Hadamard, and Kronecker products. *Journal of Mathematical Psychology*, 29(4):474–492.

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.