Named Tensor Notation

David Chiang
University of Notre Dame

Sasha Rush
Cornell University

Boaz Barak
Harvard University

Version 0.2

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):

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

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

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 $\mathsf{channels}$, 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:

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

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

$$B \in \mathbb{R}^{\mathsf{height}[3]} \qquad\qquad C \in \mathbb{R}^{\mathsf{width}[3]}$$

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

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

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

2.2.2 Reductions

The same broadcasting rules apply to functions from vectors to scalars, called *reductions*. Unlike with functions on scalars, we always have to specify which axis reductions apply to, using a subscript. (This is equivalent to the `axis` argument in NumPy and `dim` in PyTorch.)

For example, we can sum over the height axis or the width axis of A:

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

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

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

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

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:

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

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

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.

3.1 Building blocks

3.1.1 Some statistics

$$\begin{aligned}
\min_{\mathsf{ax}} A &= \min\{A_{\mathsf{ax}(i)} \mid 1 \le i \le n\} \\
\max_{\mathsf{ax}} A &= \max\{A_{\mathsf{ax}(i)} \mid 1 \le i \le n\} \\
\operatorname*{norm}_{\mathsf{ax}} A &= \sqrt{\sum_{\mathsf{ax}} A^2} \\
\operatorname*{mean}_{\mathsf{ax}} A &= \frac{1}{n} \sum_{\mathsf{ax}} A \\
\operatorname*{var}_{\mathsf{ax}} A &= \frac{1}{n} \sum_{\mathsf{ax}} (A - \operatorname*{mean}_{\mathsf{ax}} A)^2.
\end{aligned}$$

The min and max operators are overloaded, as is the summation operator defined above (§2.2.2). If the operator is applied to a tensor and has an axis under it, then it's a reduction performed over the axis. But if it is applied to a set of tensors and has no axis under it, then it's an elementwise operation performed over the set.

3.1.2 Softmax and argmax

Most activation functions are elementwise operations (sigmoid, tanh, ReLU), so they are straightforward to use in our notation; the softmax, however, is interesting because it's defined as a function from vectors to vectors:

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

As with reductions, we write an axis below the softmax operator, but this axis is retained in the output.

Closely related are argmax and argmin, which we define to compute one-hot vectors with a one at the position containing the maximum or minimum value.

$$\begin{aligned}
\operatorname*{argmax}_{\mathsf{ax}} A &= \lim_{\alpha \to \infty} \operatorname*{softmax}_{\mathsf{ax}} \alpha A \\
\operatorname*{argmin}_{\mathsf{ax}} A &= \lim_{\alpha \to -\infty} \operatorname*{softmax}_{\mathsf{ax}} \alpha A.
\end{aligned}$$

3.1.3 Fully-connected layers

A feedforward neural network looks like this:

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

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

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:

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

3.1.4 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.

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

3.1.5 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.

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

3.1.6 Convolution

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

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

where

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

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

A 2-dimensional convolution:

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

where

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

3.1.7 Max pooling

$$\begin{aligned} \mathrm{MaxPool1d}_k \colon \mathbb{R}^{\mathsf{seq}[n]} &\to \mathbb{R}^{\mathsf{seq}[n/k]} \\ \mathrm{MaxPool1d}_k(X) &= \max_{\mathsf{k}} U \end{aligned}$$

where

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

$$\begin{aligned} \mathrm{MaxPool2d}_{kh,kw} \colon \mathbb{R}^{\mathsf{height}[h] \times \mathsf{width}[w]} &\to \mathbb{R}^{\mathsf{height}[h/kh] \times \mathsf{width}[w/kw]} \\ \mathrm{MaxPool2d}_{kh,hw}(X) &= \max_{\mathsf{kh},\mathsf{kw}} U \end{aligned}$$

where

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

3.1.8 Normalization layers

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

We can define a single generic normalization layer:

$$\underset{\mathsf{ax}}{\mathrm{XNorm}} \colon \mathbb{R}^{\mathsf{ax}} \to \mathbb{R}^{\mathsf{ax}}$$

$$\underset{\mathsf{ax}}{\mathrm{XNorm}}(X; \gamma, \beta, \epsilon) = \frac{X - \underset{\mathsf{ax}}{\mathrm{mean}}(X)}{\sqrt{\underset{\mathsf{ax}}{\mathrm{var}}(X) + \epsilon}} \odot \gamma + \beta$$

where

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

Now, suppose that the input has three axes:

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

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

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

3.2 Transformer

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

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

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

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

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

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

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

where LayerNorm, SelfAtt and FFN are defined below.

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

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

We defined attention in §3.1.5; the Transformer uses multi-head self-attention, in which queries, keys, and values are all computed from the same sequence.

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

where

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

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

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

3.3 LeNet

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

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

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

The convolution and pooling operations are defined as follows:

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

where

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

and

$$\text{MaxPool}^l(X) = \text{MaxPol2d}_{ph^l, ph^l}(X).$$

3.4 Other examples

3.4.1 Discrete random variables

Named axes are very helpful for working with discrete random variables, because each random variable can be represented by an axis with the same name. For

instance, if A and B are random variables, we can treat $p(\mathsf{B} \mid \mathsf{A})$ and $p(\mathsf{A})$ as tensors:

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

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

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

$$p(\mathsf{A}, \mathsf{B}) = p(\mathsf{B} \mid \mathsf{A}) \odot p(\mathsf{A}) \qquad \text{chain rule}$$

$$p(\mathsf{B}) = \sum_{\mathsf{A}} p(\mathsf{A}, \mathsf{B}) = p(\mathsf{B} \mid \mathsf{A}) \underset{\mathsf{A}}{\odot} p(\mathsf{A}) \qquad \text{marginalization}$$

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

3.4.2 Continuous bag of words

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

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

$$\text{CBOW}(X; E, W) = \underset{\mathsf{class}}{\text{softmax}}(W \underset{\mathsf{hidden}}{\odot} E \underset{\mathsf{vocab}}{\odot} X)$$

where

$$\sum_{\mathsf{vocab}} X = 1$$

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

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

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

3.4.3 Sudoku ILP

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

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

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

where

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

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

3.4.4 K-means clustering

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

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

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

we repeat the following update: Compute cluster assignments

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

then recompute the cluster centers:

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

3.4.5 Beam search

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

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

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

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

Then we repeat the following update:

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

$$H \leftarrow \operatorname*{maxk}_{\mathsf{state},\mathsf{beam}} H'$$

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

where

$$\underset{\mathsf{ax},\mathsf{k}}{\text{maxk}}\colon \mathbb{R}^{\mathsf{ax}} \to \mathbb{R}^{\mathsf{k}}$$
$$\underset{\mathsf{ax},\mathsf{k}}{\text{argmaxk}}\colon \mathbb{R}^{\mathsf{ax}} \to \{0,1\}^{\mathsf{ax},\mathsf{k}}$$

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

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

3.4.6 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:

$$\underset{\mathsf{ax1},\mathsf{ax2}}{\det}\colon F^{\mathsf{ax1}[n]\times\mathsf{ax2}[n]} \to F$$
$$\underset{\mathsf{ax1},\mathsf{ax2}}{\text{inv}}\colon F^{\mathsf{ax1}[n]\times\mathsf{ax2}[n]} \to F^{\mathsf{ax1}[n]\times\mathsf{ax2}[n]}.$$

(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).

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

where

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

4 LaTeX Macros

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

```
\usepackage{namedtensor}
```

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

The style file contains a small number of macros:

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

5 Formal Definitions

5.1 Records and shapes

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

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

An *axis* is a pair, written $\mathsf{ax}[I]$, where ax is a name and I is a set of *indices*. We deal with axes of the form $\mathsf{ax}[[n]]$ (that is, $\mathsf{ax}[\{1, \ldots, n\}]$) so frequently that we abbreviate this as $\mathsf{ax}[n]$.

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

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

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

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

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

5.3 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 $\mathcal{U}$ orthogonal to both $\mathcal{S}$ and $\mathcal{T}$, we can extend f to:

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

If f is a multary function, we can extend its arguments to larger shapes, and we don't have to extend all the arguments with the same names. We consider just the case of two arguments; three or more arguments are analogous. Let $f\colon F^{\mathcal{S}} \times G^{\mathcal{T}} \to H^{\mathcal{U}}$ be a binary function from tensors to tensors. For any shapes $\mathcal{S}'$ and $\mathcal{T}'$ that are compatible with each other and orthogonal to $\mathcal{S}$ and $\mathcal{T}$, respectively, and $\mathcal{U}' = \mathcal{S}' \cup \mathcal{T}'$ is orthogonal to $\mathcal{U}$, we can extend f to:

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

All of the tensor operations described in §2.2 can be defined in this way. For example, the contraction operator can be defined as:

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

6 Extensions

6.1 Index types

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

6.1.1 Non-integral types

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

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

6.1.2 Integers with units

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

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

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

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

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

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

6.1.3 Tuples of integers

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

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

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

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

6.2 Indexing with a tensor of indices

Contributors: Tongfei Chen and Chu-Cheng Lin

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

Advanced indexing could be derived by taking a function

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

and extending it to higher-order tensors in its second argument according to the rules in §5.3. But because that's somewhat abstract, we give a more concrete definition below.

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

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

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

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

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

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

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

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

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

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

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

which generalizes the NumPy function of the same name.

Then we can write

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

Acknowledgements

Thanks to Ekin Akyürek, Colin McDonald, Adam Poliak, Matt Post, 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.

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.