Named Tensor Notation

David Chiang Sasha Rush Tongfei Chen Chu-Cheng Lin

December 2, 2020

Contents

1 Introduction **1**

2 Informal Overview **2**

3 Examples **7**

4 Formal Definitions **13**

5 Broadcasting and Indexing **15**

6 Duality **17**

1 Introduction

Most papers about neural networks use the notation of vectors and matrices from applied linear algebra. This notation is very well-suited to talking about vector spaces, but less well-suited to talking about neural networks. Consider the following equation (Vaswani et al., 2017):

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

where Q, K, and V are sequences of query, key, and value vectors packed into matrices. Does the product $QK^\top$ sum over the sequence, or over the query/key features? We would need to know the sizes of Q, K, and V to know that it's taken over the query/key features. Is the softmax taken over the query sequence or the key sequence? The usual notation doesn't even offer a way to answer this question. With multiple attention heads, the notation becomes more complicated and leaves more questions unanswered. With multiple sentences in a minibatch, the notation becomes more complicated still, and most papers wisely leave this detail out.

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

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

As a preview, the above equation becomes

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

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

More examples of the notation are given in § 3.

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

2 Informal Overview

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

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

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

2.1 Named tensors

A *named tuple* (also known as a *record*) looks like this:

$$\{\mathsf{foo} : 2, \mathsf{bar} : 3\}.$$

The order of the elements doesn't matter:

$$\{\mathsf{foo}:2,\mathsf{bar}:3\} = \{\mathsf{bar}:3,\mathsf{foo}:2\}.$$

We use sans-serif font for names.

Then, a real *named tensor* is a function from named tuples to reals. Each of its indices has a name, and the ordering of the indices doesn't matter. For example, here is a tensor with an index named foo ranging from 1 to 2 and an index named bar ranging from 1 to 3. More succinctly, we say that the *shape* of A is $\{\mathsf{foo}:2,\mathsf{bar}:3\}$.

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

We use uppercase italic letters for variables standing for named tensors. We don't mind if you use another convention, but urge you not to use different styles for tensors and their elements. For example, if $\mathbf{A}$ is a tensor, then an element of $\mathbf{A}$ is written as $\mathbf{A}_{\mathsf{foo}:2,\mathsf{bar}:3}$ – not $A_{\mathsf{foo}:2,\mathsf{bar}:3}$ or $a_{\mathsf{foo}:2,\mathsf{bar}:3}$.

Just as the set of all size-n real vectors is written $\mathbb{R}^n$, and the set of all $m \times n$ real matrices is often written $\mathbb{R}^{m\times n}$ (which makes sense because one sometimes writes Y^X for the set of all functions from X to Y), we write $\mathbb{R}^{\mathsf{foo}:2,\mathsf{bar}:3}$ for the set of all tensors with shape $\{\mathsf{foo}:2,\mathsf{bar}:3\}$.

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

$$A_{\mathsf{foo}:1} = \begin{matrix}\overset{\mathsf{bar}}{}\\ \begin{bmatrix}3 & 1 & 4\end{bmatrix}\end{matrix}$$

$$A_{\mathsf{bar}:3} = \begin{matrix}\overset{\mathsf{foo}}{}\\ \begin{bmatrix}4 & 9\end{bmatrix}\end{matrix}.$$

2.2 Named tensor operations

2.2.1 Elementwise operations

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

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

More elementwise unary operations:

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

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

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

then

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

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

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

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

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

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

More elementwise binary operations:

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

2.2.2 Reductions

The same rules for alignment and broadcasting apply to functions that take tensor as arguments or return tensors. The gory details are in §4.3, but we present the most important subcases here. The first is *reductions*, which are

functions from vectors to scalars. Unlike with functions on scalars, we always have to specify which index these functions apply to, using a subscript. (This is equivalent to the `axis` argument in NumPy and `dim` in PyTorch.)

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

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

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

More reductions: If A has shape $\{\mathsf{i} : X, \ldots\}$, then

$$\begin{aligned}
\sum_{\mathsf{i}} A &= \sum_{x \in X} A_{\mathsf{i}:x} \\
\operatorname*{norm}_{\mathsf{i}} A &= \sqrt{\sum_{\mathsf{i}} A^2} \\
\min_{\mathsf{i}} A &= \min_{x \in X} A_{\mathsf{i}:x} \\
\max_{\mathsf{i}} A &= \max_{x \in X} A_{\mathsf{i}:x} \\
\operatorname*{mean}_{\mathsf{i}} A &= \frac{A}{\sum_{\mathsf{i}} A} \\
\operatorname*{var}_{\mathsf{i}} A &= \sum_{\mathsf{i}} (A - \operatorname*{mean}_{\mathsf{i}} A)^2
\end{aligned}$$

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

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

2.2.3 Contraction

The vector dot product (inner product) is a function from *two* vectors to a scalar, which generalizes to named tensors to give the ubiquitous *contraction* operator:

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

This operator can also be used for matrix-vector or matrix-matrix multiplication:

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

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

However, note that (like vector dot-product, but unlike matrix multiplication) $\cdot_{\mathsf{i}}$ is commutative, but not associative! Specifically, if

$$A \in \mathbb{R}^{\mathsf{i}:m}$$
$$B \in \mathbb{R}^{\mathsf{i}:m,\mathsf{j}:n}$$
$$C \in \mathbb{R}^{\mathsf{i}:m,\mathsf{j}:n}$$

then $(A \cdot_{\mathsf{i}} B) \cdot_{\mathsf{j}} C$ and $A \cdot_{\mathsf{i}} (B \cdot_{\mathsf{j}} C)$ don't even have the same shape.

2.2.4 Vectors to vectors

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

$$\underset{\mathsf{i}}{\mathrm{softmax}}\, A = \frac{\exp A}{\sum_{\mathsf{i}} \exp A}$$

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

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

Concatenation combines two vectors into one:

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

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

2.2.5 Matrices

Finally, we briefly consider functions on matrices, for which you have to give *two* index names (and the order in general matters). Let A be a named tensor with shape $\{\mathsf{i}:2, \mathsf{j}:2, \mathsf{k}:2\}$:

$$A_{\mathsf{i}:1} = \begin{matrix} \\ \mathsf{j} \end{matrix} \overset{\mathsf{k}}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}$$

$$A_{\mathsf{i}:2} = \begin{matrix} \\ \mathsf{j} \end{matrix} \overset{\mathsf{k}}{\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}}$$

$$\det_{\mathsf{j},\mathsf{k}} A = \overset{\mathsf{i}}{\begin{bmatrix} \det \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & \det \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \end{bmatrix}}$$

$$\det_{\mathsf{k},\mathsf{j}} A = \overset{\mathsf{i}}{\begin{bmatrix} \det \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} & \det \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix} \end{bmatrix}}$$

$$\det_{\mathsf{i},\mathsf{j}} A = \overset{\mathsf{k}}{\begin{bmatrix} \det \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} & \det \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix} \end{bmatrix}}$$

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

3 Examples

3.1 Attention

Let d_k and d_v be positive integers, and let n and n' be the input and output sequence length. Define a function

$$\mathrm{Att}\colon \mathbb{R}^{\mathsf{time}':n',\mathsf{key}:d_k} \times \mathbb{R}^{\mathsf{time}:n,\mathsf{key}:d_k} \times \mathbb{R}^{\mathsf{time}:n,\mathsf{val}:d_v} \to \mathbb{R}^{\mathsf{time}':n',\mathsf{val}:d_v}$$

$$\mathrm{Att}(Q,K,V) = \operatorname*{softmax}_{\mathsf{time}}\left(\frac{Q \cdot_{\mathsf{key}} K}{\sqrt{d_k}}\right) \cdot_{\mathsf{time}} V$$

In self-attention, Q, K, and V are all computed from the same sequence. Let d_{model} be a positive integer. The parameters are:

$$W^Q \in \mathbb{R}^{\mathsf{emb}:d_{\mathrm{model}},\mathsf{key}:d_k}$$
$$W^K \in \mathbb{R}^{\mathsf{emb}:d_{\mathrm{model}},\mathsf{key}:d_k}$$
$$W^V \in \mathbb{R}^{\mathsf{emb}:d_{\mathrm{model}},\mathsf{val}:d_v}$$
$$W^O \in \mathbb{R}^{\mathsf{val}:d_v,\mathsf{emb}:d_{\mathrm{model}}}$$

Then define

$$\text{SelfAtt}\colon \mathbb{R}^{\mathsf{time}:n,\mathsf{emb}:d_{\text{model}}} \to \mathbb{R}^{\mathsf{time}:n,\mathsf{emb}:d_{\text{model}}}$$
$$\text{SelfAtt}(X; W^Q, W^K, W^V, W^O) = W^O \underset{\mathsf{val}}{\cdot} [\text{Att}(Q,K,V)]_{\mathsf{time}' \to \mathsf{time}}$$

where

$$\begin{aligned} Q &= W^Q \underset{\mathsf{emb}}{\cdot} [X]_{\mathsf{time} \to \mathsf{time}'} \\ K &= W^K \underset{\mathsf{emb}}{\cdot} X \\ V &= W^V \underset{\mathsf{emb}}{\cdot} X. \end{aligned}$$

To change this to multi-head self-attention with h attention heads, simply redefine

$$\begin{aligned} W^Q &\in \mathbb{R}^{\mathsf{head}:h,\mathsf{emb}:d_{\text{model}},\mathsf{key}:d_k} \\ W^K &\in \mathbb{R}^{\mathsf{head}:h,\mathsf{emb}:d_{\text{model}},\mathsf{key}:d_k} \\ W^V &\in \mathbb{R}^{\mathsf{head}:h,\mathsf{emb}:d_{\text{model}},\mathsf{val}:d_v} \\ W^O &\in \mathbb{R}^{\mathsf{head}:h,\mathsf{val}:d_v,\mathsf{emb}:d_{\text{model}}} \end{aligned}$$

and define

$$\text{MultiSelfAtt}\colon \mathbb{R}^{\mathsf{time}:n,\mathsf{emb}:d_{\text{model}}} \to \mathbb{R}^{\mathsf{time}:n,\mathsf{emb}:d_{\text{model}}}$$
$$\text{MultiSelfAtt}(X; W^Q, W^K, W^V, W^O) = \sum_{\mathsf{head}} \text{SelfAtt}(X; W^Q, W^K, W^V, W^O).$$

3.2 RNN

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

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

The renaming is necessary because our notation doesn't provide a one-step way to apply a linear transformation (A) to one index and put the result in the same index. For possible solutions, see §6.

3.3 Fully-Connected Layers

Fully-connected layers are bit more verbose, but make more explicit which parameters connect which layers.

$$
\begin{aligned}
V &\in \mathbb{R}^{\mathsf{output}:o,\mathsf{hidden}:h} & c &\in \mathbb{R}^{\mathsf{output}:o} \\
W &\in \mathbb{R}^{\mathsf{hidden}:h,\mathsf{in}:i} & b &\in \mathbb{R}^{\mathsf{hidden}:h} \\
X &\in \mathbb{R}^{\mathsf{batch}:b,\mathsf{in}:i} \\
Y &= \sigma\left(W \underset{\mathsf{in}}{\cdot} X + b\right) \\
Z &= \sigma\left(V \underset{\mathsf{hidden}}{\cdot} Y + c\right)
\end{aligned}
$$

3.4 Deep Learning Norms

These three functions are often informally described using the same equation, but they each correspond to very different functions. They differ by which axes are normalized.

Batch Norm

$$
\begin{aligned}
X &\in \mathbb{R}^{\mathsf{batch}:b,\mathsf{channel}:c,\mathsf{hidden}:h} \\
\gamma, \beta &\in \mathbb{R}^{\mathsf{batch}:b} \\
\mathrm{batchnorm}(X;\gamma,\beta) &= \frac{X - \mathrm{mean}_{\mathsf{batch}}(X)}{\sqrt{\mathrm{var}_{\mathsf{batch}}(X) + \epsilon}} \odot \gamma + \beta
\end{aligned}
$$

Instance Norm

$$
\begin{aligned}
X &\in \mathbb{R}^{\mathsf{batch}:b,\mathsf{channel}:c,\mathsf{hidden}:h} \\
\gamma, \beta &\in \mathbb{R}^{\mathsf{hidden}:h} \\
\mathrm{instancenorm}(X;\gamma,\beta) &= \frac{X - \mathrm{mean}_{\mathsf{hidden}}(X)}{\sqrt{\mathrm{var}_{\mathsf{hidden}}(X) + \epsilon}} \odot \gamma + \beta
\end{aligned}
$$

Layer Norm

$$
\begin{aligned}
X &\in \mathbb{R}^{\mathsf{batch}:b,\mathsf{channel}:c,\mathsf{hidden}:h} \\
\gamma, \beta &\in \mathbb{R}^{\mathsf{channel}:c,\mathsf{hidden}:h} \\
\mathrm{layernorm}(X;\gamma,\beta) &= \frac{X - \mathrm{mean}_{\mathsf{hidden},\mathsf{channel}}(X)}{\sqrt{\mathrm{var}_{\mathsf{hidden},\mathsf{channel}}(X) + \epsilon}} \odot \gamma + \beta
\end{aligned}
$$

3.5 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.

$$X \in \{0,1\}^{\mathsf{time}:t,\mathsf{vocab}:v} \qquad \sum_{\mathsf{vocab}} X = 1$$

$$E \in \mathbb{R}^{\mathsf{vocab}:v,\mathsf{hidden}:h}$$

$$W \in \mathbb{R}^{\mathsf{class}:c,\mathsf{hidden}:h}$$

$$\mathrm{cbow}(X; E, W) = \underset{\mathsf{class}}{\mathrm{softmax}}(W \underset{\mathsf{hidden}}{\cdot} E \underset{\mathsf{vocab}}{\cdot} X)$$

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

3.6 Bayes' Rule

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

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

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

Then Bayes' rule is just:

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

3.7 Sudoku ILP

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

$$X \in \{0,1\}^{\mathsf{row}:9,\mathsf{col}:9,\mathsf{assign}:9}$$

$$\mathrm{check}(X) = \left(\sum_{\mathsf{assign}} Y = \sum_{\mathsf{Row},\mathsf{row}} Y = \sum_{\mathsf{Col},\mathsf{col}} Y = \sum_{\mathsf{row},\mathsf{col}} Y = 1 \right)$$

$$Y \in \{0,1\}^{\mathsf{Row}:3,\mathsf{Col}:3,\mathsf{row}:3,\mathsf{col}:3,\mathsf{assign}:9}$$

$$Y_{\mathsf{Row}:r,\mathsf{row}:r',\mathsf{Col}:c,\mathsf{col}:c'} = X_{\mathsf{row}:r\times 3+r'-1,\mathsf{col}:c\times 3+c'-1},$$

3.8 Max Pooling

Max pooling used in image recognition takes a similar form as the Sudoku example.

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

3.9 1D Convolution

1D Convolution can be easily written by unrolling a tensor and then applying a standard dot product.

$$\begin{aligned} X &\in \mathbb{R}^{\mathsf{channel}:c,\mathsf{time}:t} \\ W &\in \mathbb{R}^{\mathsf{out\_channel}:c',\mathsf{channel}:c,\mathsf{kw}:k} \\ \text{conv1d}(X, W) &= W \underset{\mathsf{channel},\mathsf{kw}}{\cdot} U \\ U &\in \mathbb{R}^{\mathsf{channel}:c,\mathsf{time}:t-k+1,\mathsf{kw}:k} \\ U_{\mathsf{time}:i,\mathsf{kw}:\mathsf{j}} &= X_{\mathsf{time}:i+j-1} \end{aligned}$$

3.10 K-Means Clustering

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

$$\begin{aligned} X &\in \mathbb{R}^{\mathsf{batch}:b,\mathsf{dim}:k} \\ C &\in \mathbb{R}^{\mathsf{cluster}:c,\mathsf{dim}:k} \end{aligned}$$

we compute cluster assignments

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

then we recompute the cluster centers:

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

3.11 Beam Search

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

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

3.12 Multivariate Normal

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

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

3.13 Attention with Causal Masking

When the Transformer is used for generation, it is necessary to have an additional mask to ensure the model does not look at future words. This can be included in the attention definition with clear names.

$$Q \in \mathbb{R}^{\mathsf{key}:d_v,\mathsf{time}':n}$$
$$K \in \mathbb{R}^{\mathsf{head}:h,\mathsf{key}:d_k,\mathsf{time}:n}$$
$$V \in \mathbb{R}^{\mathsf{head}:h,\mathsf{val}:d_v,\mathsf{time}:n}$$
$$\text{attention}(Q,K,V) = \underset{\mathsf{time}}{\text{softmax}}\left(\frac{Q \underset{\mathsf{key}}{\cdot} K}{\sqrt{d_k}} + M\right) \underset{\mathsf{time}}{\cdot} V$$
$$M \in \mathbb{R}^{\mathsf{time}:n,\mathsf{time}':n}$$
$$M_{\mathsf{time}:i,\mathsf{time}':j} = \begin{cases} 0 & i \le j \\ -\infty & \text{otherwise} \end{cases}$$

3.14 Full Examples: Transformer and LeNet

As further proof of concept, we have written the full models for Transformer (`https://namedtensor.github.io/transformer.html`) and LeNet (`https://namedtensor.github.io/convnet.html`).

4 Formal Definitions

4.1 Named tuples

A *named tuple* is a set of pairs, written as $\{\mathsf{i}_1 : x_1, \ldots, \mathsf{i}_r : x_r\}$, where $\mathsf{i}_1, \ldots \mathsf{i}_r$ are pairwise distinct *names*. We write both names and variables ranging over names using sans-serif font.

If t is a named tuple, we write $\operatorname{dom} t$ for the set $\{\mathsf{i} \mid (\mathsf{i} : x) \in t \text{ for some } x\}$. If $\mathsf{i} \in \operatorname{dom} t$, we write $t.\mathsf{i}$ for the unique x such that $(\mathsf{i} : x) \in t$. We write the empty named tuple as $\emptyset$.

We define a partial ordering $\sqsubseteq$ on named tuples: $t_1 \sqsubseteq t_2$ iff for all i, x, $(\mathsf{i} : x) \in t_1$ implies $(\mathsf{i} : x) \in t_2$. Then $t_1 \sqcap t_2$ is the greatest lower bound of t_1 and t_2, and $t_1 \sqcup t_2$ is their least upper bound.

A named tuple $\{\mathsf{i}_1 : X_1, \ldots, \mathsf{i}_r : X_r\}$ where $X_1, \ldots, X_r$ are sets is called a *shape*, which we will often use to yield a set of named tuples:

$$\operatorname{ind}\{\mathsf{i}_1 : X_1, \ldots, \mathsf{i}_r : X_r\} = \{\{\mathsf{i}_1 : x_1, \ldots, \mathsf{i}_r : x_r\} \mid x_1 \in X_1, \ldots, x_r \in X_r\}\,.$$

If $t \in \operatorname{ind} \mathcal{T}$ and $\mathcal{S} \sqsubseteq \mathcal{T}$, then we write $t|_{\mathcal{S}}$ for the named tuple $\{(\mathsf{i} : x) \in t \mid \mathsf{i} \in \operatorname{dom} S\}$.

4.2 Named tensors

Let $[n] = \{1, \ldots, n\}$. We deal with shapes of the form $\{\mathsf{i}_1 : [n_1], \ldots, \mathsf{i}_r : [n_r]\}$ so frequently that we define the shorthand $\{\mathsf{i}_1 : n_1, \ldots, \mathsf{i}_r : n_r\}$.

Let F be a field and let $\mathcal{T}$ be a shape. Then a *named tensor over F with shape $\mathcal{T}$* is a mapping from $\operatorname{ind}\mathcal{T}$ to F. We write the set of all named tensors with shape $\mathcal{T}$ as $F^{\mathcal{T}}$. To avoid clutter, in place of $F^{\{\mathsf{i}_1:X_1,\ldots,\mathsf{i}_r:X_r\}}$, we usually write $F^{\mathsf{i}_1:X_1,\ldots,\mathsf{i}_r:X_r}$.

We don't make any distinction between a scalar (an element of F) and a named tensor with empty shape (an element of $F^{\emptyset}$).

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

We also allow partial indices. Let $\mathcal{U}$ be a shape such that $\mathcal{U} = \mathcal{S} \sqcup \mathcal{T}$ and $\mathcal{S} \sqcap \mathcal{T} = \emptyset$. If A is a tensor with shape $\mathcal{S}$ and $s \in \operatorname{ind}\mathcal{S}$, then we define A_s to be the named tensor with shape $\mathcal{T}$ such that, for any $t \in \operatorname{ind}\mathcal{T}$,

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

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

4.3 Extending functions to named tensors

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

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

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

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

$$\begin{aligned} \mathcal{S} \sqcap \mathcal{S}' &= \emptyset \\ \mathcal{T} \sqcap \mathcal{T}' &= \emptyset \\ \mathcal{U} \sqcap \mathcal{U}' &= \emptyset \end{aligned}$$

we can extend f to:

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

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

$$\underset{\mathsf{i}}{\cdot} : F^{\mathsf{i}:n} \times F^{\mathsf{i}:n} \to F$$

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

5 Broadcasting and Indexing

Broadcasting is extremely common in array-based programming, despite its being implicit and prone to errors.[1] Under the named tensor notations, we can develop an alternative formulation of broadcasting and indexing.

We say that named tensor shapes $\mathcal{S}$ is *broadcastable* to $\mathcal{T}$ if $\mathcal{S} \sqsubseteq \mathcal{T}$. This is compatible with the NumPy semantics except that they allow dimensions with size 1.

Consider the following operations between tensors $A \in \mathbb{R}^{\mathsf{i}:m}$ and $B \in \mathbb{R}^{\mathsf{j}:n}$. Sometimes we desire the following computation:

$$C_{\mathsf{i}:i,\mathsf{j}:j} = A_{\mathsf{i}:i} + B_{\mathsf{j}:j} \ .$$

Normally, we need to expand the dimensionalities of both vectors by 1, and then add them. A question here is: can we broadcast a tuple of named tensors to the same shape? In fact we can.

$$\text{broadcast} : X_1^{\mathcal{S}_1} \times \cdots \times X_n^{\mathcal{S}_n} \to X_1^{\mathcal{T}} \times \cdots \times X_n^{\mathcal{T}}, \quad \mathcal{T} = \bigsqcup_{i=1}^{n} \mathcal{S}_i$$

if all axes with shared names match. We'll use this to develop a formalism of advanced indexing under named tensor notations.

Advanced indexing A common operation in array-based programming is *advanced indexing*, whose semantics is vague[2] and cannot be easily reasoned over. Given a tensor $A \in X^{\mathsf{i}_1:n_1,\cdots,\mathsf{i}_D:n_D}$, in NumPy (or PyTorch), A can be indexed in the form of $A[I^1, \cdots, I^D]$, where each *indexer* $I^{d \in [D]}$ can be of the following three types:

- An index $I^d \in [n_d]$;
- A subset of indices $I^d \subseteq [n_d]$ (a special case is ":", i.e. $I^d = [n_d]$);
- Another tensor with integer elements $I^d \in [n_d]^{\mathcal{T}}$ with shape $\mathcal{T}$.

[1] This is, in my opinion, a glaring negligence of the *Zen of Python*: "Explicit is better than implicit".

[2] `https://numpy.org/doc/stable/reference/arrays.indexing.html`.

We first start our discussion where all indexers $I^{d\in[D]}$ to the indexee $A \in X^{\mathcal{S}}$ are tensors in $[n_d]^{\mathcal{T}}$ with uniform shape $\mathcal{T} = \{\mathsf{j}_1 : m_1, \cdots, \mathsf{j}_E : m_E\}$. The advanced indexing operation is defined as

$$\begin{aligned} \text{advIdx} : X^{\mathcal{S}} \times ([n_1]^{\mathcal{T}} \times \cdots \times [n_D]^{\mathcal{T}}) &\to X^{\mathcal{T}} \\ B = A[I^1, \cdots, I^d] = \text{advIdx}(A, (I^1, \cdots, I^d)) &\in X^{\mathcal{T}} \\ B_{\mathsf{j}_1:y_1,\cdots,\mathsf{j}_E:y_E} &= A_{\mathsf{i}_1:I^1_{\mathsf{j}_1:y_1,\cdots,\mathsf{j}_E:y_E},\cdots,\mathsf{i}_D:I^D_{\mathsf{j}_1:y_1,\cdots,\mathsf{j}_E:y_E}} \end{aligned}$$

How do we fit the three types of indexers into this definition? We view each indexer I^d as a named tensor with the following shape:

- An index $I^d \in [n_d]$: We view $I^d \in [n_d]^{\emptyset}$;
- A subset of indices $I^d \subseteq [n_d]$: We view this as a 1-dimensional tensor with name i_d, hence $I^d \in [n_d]^{\mathsf{i}_d:|I^d|}$;
- Another tensor with integer elements $I^d \in [n_d]^{\mathcal{T}}$ with shape $\mathcal{T}$.

Now the more lenient form of advanced indexing as in the semantics of NumPy can be described using the broadcast operator we elaborated above:

$$\begin{aligned} \text{advIdx}' : X^{\mathcal{S}} \times ([n_1]^{\mathcal{T}_1} \times \cdots \times [n_D]^{\mathcal{T}_D}) &\to X^{\bigsqcup_{i=1}^{D} \mathcal{T}_i} \\ B = A[I^1, \cdots, I^d] = \text{advIdx}(A, \text{broadcast}(I^1, \cdots, I^d)) &\in X^{\mathcal{T}} \end{aligned}$$

Let's see a concrete example in natural language processing. Given a batch of encoded sentences using a contextualizer like BERT, we get a tensor $X \in \mathbb{R}^{\mathsf{batch}:B,\mathsf{sentLen}:N,\mathsf{emb}:E}$. For each sentence, we would like to take out the encodings of a specific span $[l_b, r_b)$ for each sentence $b \in [B]$ in the batch, resulting in a tensor $Y \in \mathbb{R}^{\mathsf{batch}:B,\mathsf{spanLen}:M,\mathsf{emb}:E}$.

We create a indexer for the $\mathsf{sentLen}$ axis: $I_{\mathsf{sentLen}} \in [N]^{\mathsf{batch}:B,\mathsf{spanLen}:M}$ that selects the desired encodings of tokens. The advanced indexing expression $X[:, I, :]$ can be viewed under our formulation as

$$\text{advIdx}'(X, (I_{\mathsf{batch}}, I_{\mathsf{sentLen}}, I_{\mathsf{emb}})) \ .$$

where

$$\begin{aligned} I_{\mathsf{batch}} &\in [B]^{\mathsf{batch}:B} \ ; \\ I_{\mathsf{sentLen}} &\in [N]^{\mathsf{batch}:B,\mathsf{spanLen}:M} \ ; \\ I_{\mathsf{emb}} &\in [E]^{\mathsf{emb}:E} \ . \end{aligned}$$

Based on our broadcast function, the three indexers when broadcast together has shape $\{\mathsf{batch} : B, \mathsf{spanLen} : M, \mathsf{emb} : E\}$ just as we desired.

6 Duality

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

In §3.2 we saw that defining an RNN requires renaming of indices, because a linear transformation must map one index to another index; if we want to map an index to itself, we need to use renaming.

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

6.1 Contracting two names

We define a version of the contraction operator that can contract two indices with different names. If $\mathsf{i} \in \operatorname{dom} \mathcal{A}$ and $\mathsf{j} \in \operatorname{dom} \mathcal{B}$ and $\mathcal{A}.\mathsf{i} = \mathcal{B}.\mathsf{j} = X$, then we define

$$A \underset{\mathsf{i}|\mathsf{j}}{\cdot} B = \sum_{x \in X} A_{\mathsf{i}:x} \, B_{\mathsf{j}:x}$$

For example, the RNN would look like this.

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

6.2 Starred index names

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

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

In general, if $\mathsf{i}* \in \operatorname{dom}\mathcal{A}$ and $\mathsf{i} \in \operatorname{dom}\mathcal{B}$ and $\mathcal{A}.\mathsf{i}* = \mathcal{B}.\mathsf{i} = X$, then we define

$$A \underset{\mathsf{i}}{\cdot} B = \sum_{x \in X} A_{\mathsf{i}*:x}\, B_{\mathsf{i}:x}$$

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

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

6.3 Natural contraction

Tensor contraction is variously termed as `einsum` (Einstein summation) in NumPy or `tensordot` in various integer-indexed tensor libraries. These notations require a list of index pairs to designate which axes to contract (e.g. `einsum([a, b], axes=[(1, 2), (2, 3)])`), which is pretty unreadable to a programmer. We consider a special case here.

Consider two tensors $A \in \mathbb{R}^{\mathcal{S}}$ and $B \in \mathbb{R}^{\mathcal{T}}$. We define the *symmetric difference* of two shapes as

$$\mathcal{S} \triangle \mathcal{T} = \{(\mathsf{i} : x) \mid \mathsf{i} \in \operatorname{dom}\mathcal{S} \triangle \operatorname{dom}\mathcal{T}\} \ ,$$

where $\mathcal{S}|_{\mathcal{S}\sqcap\mathcal{T}} = \mathcal{T}|_{\mathcal{S}\sqcap\mathcal{T}}$, i.e., axes with the same name matches. For example, the shape of matrix multiplication can be expressed using the symmetric difference operator:

$$\{\mathsf{i} : m, \mathsf{j} : n\} \triangle \{\mathsf{j} : n, \mathsf{k} : p\} = \{\mathsf{i} : m, \mathsf{k} : p\} \ . \tag{1}$$

Now we can introduce the notion of *natural contraction* (Chen, 2017). We define an operator

$$\bowtie \ : \mathbb{R}^{\mathcal{S}} \times \mathbb{R}^{\mathcal{T}} \to \mathcal{R}^{\mathcal{S}\triangle\mathcal{T}},$$

where

$$A \bowtie B \triangleq \sum_{x_1 \in X_1} \cdots \sum_{x_D \in X_D} A_{\mathsf{i}_1:x_1,\cdots,\mathsf{i}_D:x_D} \cdot B_{\mathsf{i}_1:x_1,\cdots,\mathsf{i}_D:x_D} \quad \forall(\mathsf{i}_d : X_d) \in \mathcal{S}\triangle\mathcal{T} \ .$$

This says, given two tensors A and B, contract all axes where the names match, and retain all other axes. Clearly, this subsumes various operations in linear algebra:

Inner product	$\mathbf{a} \in \mathbb{R}^{\mathsf{i}:n}$, $\mathbf{b} \in \mathbb{R}^{\mathsf{i}:n}$	$\mathbf{a} \cdot \mathbf{b} = \mathbf{a} \bowtie \mathbf{b} \in \mathbb{R}^{\emptyset}$
Matrix product	$\mathbf{A} \in \mathbb{R}^{\mathsf{i}:m,\mathsf{j}:n}$, $\mathbf{B} \in \mathbb{R}^{\mathsf{j}:n,\mathsf{k}:p}$	$\mathbf{AB} = \mathbf{A} \bowtie \mathbf{B} \in \mathbb{R}^{\mathsf{i}:m,\mathsf{k}:p}$
Tensor product	$\mathbf{A} \in \mathbb{R}^{\mathcal{S}}$, $\mathbf{B} \in \mathbb{R}^{\mathcal{T}}$ (if $\mathcal{S} \sqcap \mathcal{T} = \emptyset$)	$\mathbf{A} \otimes \mathbf{B} = \mathbf{A} \bowtie \mathbf{B} \in \mathbb{R}^{\mathcal{S}\sqcup\mathcal{T}}$

Belief propagation under named tensor notation The natural contraction operator is a natural fit for describing the belief propagation algorithm in graphical models. Consider a function g over a set of variables $\mathcal{X} = \{x_1, \cdots, x_N\}$ where $x_i \in X_i$:

$$g(\mathcal{X}) = \prod_{j=1}^{M} f_j(S_j)$$

where $S_j \subseteq \mathcal{X}$ is a subset of variables in $\mathcal{X}$. The corresponding factor graph $G = (\mathcal{X}, F, E)$ is a bipartite graph where

$\mathcal{X} = \{x_1, \cdots, x_N\}$	Set of variables
$F = \{f_1, \cdots, f_M\}$	Set of potential functions
$E = \{(x_i, f_j) \mid x_i \in S_j\}$	Set of undirected edges depending on factorization

We assign each variable x_i with a distinct *name* i_i. The potential function f_j over variables S_j can now be viewed as a *named tensor* with the following shape:

$$\begin{aligned} &\mathcal{S}_j = \{(\mathsf{i}_i : X_i)\} \quad \forall x_i \in S_j \ ; \\ &f_j \in \mathbb{R}^{\mathcal{S}_j} \ . \end{aligned}$$

Additionally, messages being passed in the belief propagation algorithm, along with their computation in the sum-product algorithm, can also be described succinctly using named tensor notations:

$$M_{x_i \to f_j} \quad = \quad \bigodot_{f_k \in \mathrm{Nb}(x_i) \setminus \{f_j\}} M_{f_k \to x_i} \quad \in \quad \mathbb{R}^{\mathsf{i}_i : X_i}$$

$$M_{f_j \to x_i} \quad = \quad f_j \bowtie \bigbowtie_{x_k \in \mathrm{Nb}(f_j) \setminus \{x_i\}} M_{x_i \to f_j} \quad \in \quad \mathbb{R}^{\mathsf{i}_i : X_i}$$

where $\mathrm{Nb}(\cdot)$ is the set of neighboring nodes in the factor graph G.

6.4 Named and numbered indices

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

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

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

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

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

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

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

We also get the renaming operation for free: $A_{\mathsf{i}\to\mathsf{j}} = [A_{\mathsf{i}}]_{\to\mathsf{j}}$ renames index i to j.

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

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

or equivalently,

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

Attention:

$$\begin{aligned} \mathrm{Att}\colon \mathbb{R}^{\mathsf{time}':n',\mathsf{key}:d_k} \times \mathbb{R}^{\mathsf{time}:n,\mathsf{key}:d_k} \times \mathbb{R}^{\mathsf{time}:n,\mathsf{val}:d_v} &\to \mathbb{R}^{\mathsf{time}':n',\mathsf{val}:d_v} \\ \mathrm{Att}(Q,K,V) = \mathrm{softmax}\left[\frac{Q_{\mathsf{key}} \cdot K_{\mathsf{key}}}{\sqrt{d_k}}\right]_{\mathsf{time}} &\cdot V_{\mathsf{time}} \end{aligned}$$

Multivariate normal distribution:

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

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

Acknowledgements

Thanks to Ekin Akyürek, Colin McDonald, Chung-chieh Shan, and Nishant Sinha for their input to this document (or the ideas in it).

References

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

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

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

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

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

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

Torch Contributors. 2019. Named tensors. PyTorch documentation.

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