Common Functions on Tensors

Parakram Majumdar

September 2, 2018

1 Introduction

This document describes some of the standard tensor operations that are relevant for machine learning. The rigorous mathematician may note that this document uses the term *tensor* merely as a substitute for *multi-dimensional arrays*.

2 Definitions

- A *tensor* T of *dimensionality* $d_1, d_2, ..., d_o$ is a multi-dimensional array of real numbers in $R^{d_1 \times d_2 \times ... \times d_o}$.
- In the above definition, the number of dimensions o is also called the *order* of T.
- The number of elements in T, i.e., $d_1 \times d_2 \times ... \times d_o$ is called the size of T.
- For brevity, we introduce the following notations:

$$\begin{aligned} \overrightarrow{d_n} &\equiv d_1, d_2, ..., d_n \\ \Pi d_n &\equiv d_1 \times d_2 \times ... \times d_n \end{aligned}$$

- A single argument *tensor function*

$$f : \mathbb{R}^{\Pi a_m} \rightarrow \mathbb{R}^{\Pi b_n}$$

  is a function that maps a tensor to another tensor.

- The *gradient* of the above mentioned f at some point $x \in \mathbb{R}^{\Pi a_m}$ is a tensor

$$\nabla f(x) \in \mathbb{R}^{\Pi a_m \times \Pi b_n}$$

  that gives the rate of change of each element of $f(x)$ with respect to each element of x.

$$\frac{\partial f(x)[\overrightarrow{j_n}]}{\partial x[\overrightarrow{i_m}]} = \nabla f(x)[\overrightarrow{i_m}, \overrightarrow{j_n}]$$

- A multiple argument tensor function takes multiple input tensors, and hence, has a separate gradient tensor with respect to each input. Note that, in general, it is impossible to combine all these gradients into a single tensor, since the various inputs may have different dimensionalities.

3 Grouping of co-ordinates

The tensors are implemented as a row-major one dimentional array with zero based indexing. For example, the three dimensional tensor $X \in \mathbb{R}^{2\times3\times4}$ is stored as a one dimensional array of size $2 \times 3 \times 4 = 24$. The elements may be accessed by the formula:

$$X[i, j, k] = X[i \times 3 \times 4 + j \times 4 + k]$$

where $i \in \{0, 1\}$, $j \in \{0, 1, 2\}$ and $k \in \{0, 1, 2, 3\}$.

An interesting consequence of this is that consecutive dimensions may be squashed together without affecting the contents of the tensor. For example, $X \in \mathbb{R}^{2\times3\times4}$ can be seen to be in $\mathbb{R}^{2\times12}$, $\mathbb{R}^{6\times4}$ and $\mathbb{R}^{24}$, as long as the correct formulae are used to interconvert the coordinates between these various forms.

To further clarify, we temporarily introduce the notation of left superscripting the dimensionality of X to denote which format it is in. Thus:

$$\begin{aligned} X[i, j, k] &= {}^{2\times3\times4}X[i, j, k] \\ &= {}^{2\times12}X[i, 4j + k] \\ &= {}^{6\times4}X[3i + j, k] \\ &= {}^{24}X[12i + 4j + k] \end{aligned}$$

We now state again the above equivalence in even crisper notation by introducing the grouping of co-ordinates:

$$\begin{aligned} X[i, j, k] &= X[[i], [j], [k]] \\ &= X[[i, j], k] \\ &= X[i, [j, k]] \\ &= X[[i, j, k]] \end{aligned}$$

4 Identity

The identity tensor function $\mathbb{I}$ maps each tensor to itself, and hence, is usually not useful in an efficient computation graph. However, it is mentioned here as an example of how the gradient of simple tensor functions can be remarkably sparse.

Consider an input $x \in \mathbb{R}^{\Pi a_n}$. The gradient of $\mathbb{I}$ would then be

$$\begin{aligned}
\nabla\mathbb{I}(x)[i_1, i_2, ..., i_n, j_1, j_2, ..., j_n] &= \frac{\partial(\mathbb{I}(x)[j_1, j_2, ..., j_n])}{\partial x[i_1, i_2, ..., i_n]} \\
&= \frac{\partial(\mathbb{I}(x)[\overrightarrow{j}])}{\partial x[\overrightarrow{i}]} \\
&= \frac{\partial x[\overrightarrow{j}]}{\partial x[\overrightarrow{i}]} \\
&= \begin{cases} 1, & \text{if } \overrightarrow{i} == \overrightarrow{j} \\ 0, & \text{otherwise} \end{cases}
\end{aligned}$$

Thus, $\nabla\mathbb{I}(x)$ is a *diagonal* tensor with very few non-zero elements.

5 Back Propagation

A common goal in machine learning is to minimize an objective with respect to some parameters. The back propagation algorithm is a *gradient descent* algorithm implemented on a computation graph, i.e., it computes the gradient of the objective with respect to each parameter, and then shifts the parameters to the direction where the objective seems to be *descending*.

The algorithm starts off by computing the value of the each node in the computation graph, starting with the values of the nodes directly defined on the parameters, and ultimately the value of the final objective.

Then, to compute the gradient of the objective with respect to the parameters, it starts applying the *backward propagation* step on each computation node, starting from the objective, and ending at the parameters.

Thus, the algorithm uses *forward propagation* for computing the value of the objective, and then a *backward propagation* to compute the gradient.

As an example, suppose a given node f has an input tensor x. x might be a parameter to the overall graph, or it might be the output of some other node. Also, suppose f feeds into some other nodes g and h. We will demonstrate the application of the backward propagation step to f.

Assume that the objective of the graph is o. The backward propagation step on f assumes that $g^*(f(x))$ and $h^*(f(x))$ are given, and computes $f^*(x)$, where

$$g^* = \frac{\partial o}{\partial g}, h^* = \frac{\partial o}{\partial h}, f^* = \frac{\partial o}{\partial f}$$

by using the simple relationship:

$$f^* = \frac{\partial o}{\partial f} = \frac{\partial o}{\partial g} \cdot \frac{\partial g}{\partial f} + \frac{\partial o}{\partial h} \cdot \frac{\partial h}{\partial f} = g^* \cdot g' + h^* \cdot h'$$

The operator $\cdot$ in the above equation is a generalization of the *vector dot product* to tensors. We discuss this *tensor dot product* operator in a later section.

Also, note that in general, f could have more than one inputs, and could have any number of outputs, some of which might be repeated, in case f is a parameter to them more than once. These generalisations, being somewhat simple, are left to the reader.

6 Tensor Dot Product

Tensor dot product is a generalization of vector dot product and matrix multiplication. Given two tensors

$$L \in \mathbb{R}^{\Pi a_m \times \Pi b_n}$$

$$R \in \mathbb{R}^{\Pi b_n \times \Pi c_o}$$

the *tensor dot product of L with R on n dimensions*

$$L \cdot_n R = M \in \mathbb{R}^{\Pi a_m \times \Pi c_o}$$

is computed as [1]:

$$M[\overrightarrow{a}, \overrightarrow{c}] = \sum_{\overrightarrow{i}=\overrightarrow{0}_n}^{\overrightarrow{i}<\overrightarrow{b}_n} L[\overrightarrow{a}, \overrightarrow{i}] \times R[\overrightarrow{i}, \overrightarrow{c}]$$

Thus, matrix multiplication and vector dot product, can be seen as tensor dot products, of matrices and vectors, respectively, on 1 dimension.

To revisit how the tensor dot product is connected with the backward propagation algorithm, suppose we have two tensor functions

$$f : \mathbb{R}^{\Pi a_m} \to \mathbb{R}^{\Pi b_n}$$

$$g : \mathbb{R}^{\Pi b_n} \to \mathbb{R}^{\Pi c_o}$$

with gradients

$$\nabla f : \mathbb{R}^{\Pi a_m} \to \mathbb{R}^{\Pi a_m \times \Pi b_n}$$
$$\nabla g : \mathbb{R}^{\Pi b_n} \to \mathbb{R}^{\Pi b_n \times \Pi c_o}$$

then the gradient of

$$g \circ f = g(f(\cdot)) : \mathbb{R}^{\Pi a_m} \to \mathbb{R}^{\Pi c_o}$$

denoted by

$$\nabla(g \circ f) : \mathbb{R}^{\Pi a_m} \to \mathbb{R}^{\Pi a_m \times \Pi c_o}$$

is computed by the *chain rule*:

$$\nabla(g \circ f)(x) = \nabla g(f(x)) \cdot_n \nabla f(x), \forall x \in \mathbb{R}^{\Pi a_m}$$

[1] The astute reader will note how the bounds of the *loop variable* $\overrightarrow{i}$ are element-wise from $[0, 0...(n \text{ times})]$ to $[b_1 - 1, b_2 - 1, ...b_n - 1]$. This is consistent with the low level storage details discussed in an earlier section.

Or in short:

$$\nabla(g \circ f) = \nabla g \cdot \nabla f$$

We now look at the gradients of the dot product product, which can of course be seen as a binary tensor, with a dimensionality in the above example of:

$$(\cdot_n) : \mathbb{R}^{\Pi a_m \times \Pi b_n} \times \mathbb{R}^{\Pi b_n \times \Pi c_o} \to \mathbb{R}^{\Pi a_m \times \Pi c_o}$$

There are of course two partial gradients, one with respect to each of the two arguments:

$$\nabla_L M = \frac{\partial M}{\partial L} \in \mathbb{R}^{\Pi a_m \times \Pi b_n \times \Pi a_m \times c_o}$$

$$\nabla_R M = \frac{\partial M}{\partial R} \in \mathbb{R}^{\Pi b_n \times \Pi c_o \times \Pi a_m \times \Pi c_o}$$

These are defined as:

$$\begin{aligned}
\nabla_L M[\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{a'}, \overrightarrow{c}] &= \frac{\partial M[\overrightarrow{a'}, \overrightarrow{c}]}{\partial L[\overrightarrow{a}, \overrightarrow{b}]} \\
&= \frac{\partial \sum_{\overrightarrow{i} \leftarrow \overrightarrow{b}} L[\overrightarrow{a'}, \overrightarrow{i}] \times R[\overrightarrow{i}, \overrightarrow{c}]}{\partial L[\overrightarrow{a}, \overrightarrow{b}]} \\
&= \begin{cases} R[\overrightarrow{b}, \overrightarrow{c}], & \text{if } \overrightarrow{a'} == \overrightarrow{a} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

$$\begin{aligned}
\nabla_R M[\overrightarrow{b}, \overrightarrow{c}, \overrightarrow{a}, \overrightarrow{c'}] &= \frac{\partial M[\overrightarrow{a}, \overrightarrow{c'}]}{\partial R[\overrightarrow{b}, \overrightarrow{c}]} \\
&= \frac{\partial \sum_{\overrightarrow{i} \leftarrow \overrightarrow{b}} L[\overrightarrow{a}, \overrightarrow{i}] \times R[\overrightarrow{i}, \overrightarrow{c'}]}{\partial R[\overrightarrow{b}, \overrightarrow{c}]} \\
&= \begin{cases} L[\overrightarrow{a}, \overrightarrow{b}], & \text{if } \overrightarrow{c'} == \overrightarrow{c} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

7 Add scalar

The core of a perceptron revolves around the formula:

$$p = w \cdot x + b\overrightarrow{u}$$

where

- p is the *output*
- w are the *weights* of the perceptron
- x is the *input*

- b is the *bias*
- u is a unit tensor created to add b to every element of p

We define the scalar addition function $\boxplus$ to encapsulate the addition of b to each element in the output

$$z \boxplus b = z + b.\overrightarrow{u}$$

Thus,

$$\begin{aligned} \frac{\partial(z \boxplus b)}{\partial z} &= \nabla\mathbb{I}(z) \\ \frac{\partial(z \boxplus b)}{\partial b} &= \nabla\mathbb{I}(z) \end{aligned}$$

8 Sigmoid

The sigmoid function on a scalar is defined as:

$$\text{sigmoid}(x) = \frac{e^x}{1+e^x}$$

The derivative is therefore given as:

$$\frac{\partial \text{sigmoid}(x)}{\partial x} = \frac{e^x}{(1+e^x)^2} = \text{sigmoid}(x)(1 - \text{sigmoid}(x))$$

We extend this to tensors as an element wise application of the scalar version

$$\text{sigmoid}(z) = \text{map}(z, \text{sigmoid}(.))$$

And therefore, for an input $z \in \mathbb{R}^{\Pi a_n}$, the gradient $\nabla\text{sigmoid}(z) \in \mathbb{R}^{\Pi a_n \times \Pi a_n}$ is defined as:

$$\nabla\text{sigmoid}(z)[\overrightarrow{i_n}, \overrightarrow{j_n}] = \begin{cases} \text{sigmoid}(z[\overrightarrow{i_n}]) \cdot (1 - \text{sigmoid}(z[\overrightarrow{i_n}])) & \text{if } \overrightarrow{i_n} == \overrightarrow{j_n} \\ 0 & \text{otherwise} \end{cases}$$