What Is Einstein Summation (Einsum)? Should I Use It?

Evan Dramko

March 2025

TL;DR: The numpy and pyTorch `einsum` functions are a simplified version of Einstein Notation in which repeated indices are summed. It is less efficient than more specific function calls like `matmul` or `dot`, but provides greater flexibility and can succinctly handle computations that do not have another convenient existing library function.

1 Introduction

As LLM coding tools become ever more prevalent, data scientists and machine learning experts are becoming increasingly exposed to the numpy and pyTorch's `einsum` function. Despite its prevalance in LLM generated code, many programmers have never heard of or used it before. Here, we will break down the function and when it should be used.

2 Where Does It Come From?

The `einsum` command is a call to a function written in Einstein Notation. As you may have guessed, it was coined by Albert Einstein, and created to represent a wide variety of summation operations concisely. While primarily derived for physics applications, a (somewhat restricted) version has been created which focused on summations over matrix and vector-valued data types. This is what numpy and pyTorch support, and is what we will cover.

3 How Does It Work?

`einsum` is actually quite intuitive and easy to use, although reading the rules can seem quite complicated. The best way to learn it is to work through the examples using the list of rules as a reference.

3.1 Rules of Einsum

Operations written using `einsum` are determined by the indices used to represent each dimension in the matrices (or tensors) of the input. It is written symbolically in the form: $A_{im_A} B_{im_B} = C_{im_C}$, and in numpy and pyTorch it is represented by:

$$C = im_A, im_B \rightarrow im_C, A, B$$

where im_A, im_B, im_c are the indices of the input tensors A, B and C respectively. The rules to writing the indices are as follows:

1. Indices that are repeated in im_A, imB and don't appear in the output are element-wise multiplied and summed over (dot product). Thus any dot-product indices must have the same length.

2. The non-summed indices between im_A and im_B should appear in im_C. [1]

Note that einsum does not explicitly consider dimensions of the matrices/tensors but rather implicitly assumes that the user has worked them out ahead of time.

3.2 Examples

3.2.1 Example 1: Vector Dot Product

Consider input vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. To compute the dot product:

$$\underbrace{\mathbf{a}_i \mathbf{b}_i}_{\text{Einstein}} = \sum_i \mathbf{a}_i \mathbf{b}_i = \mathbf{a} \cdot \mathbf{b}$$

This can be read as: *for every non-summed dimension in the output we perform a sum along axis i.* Since there is no non-summed dimension, it is implicitly the $0th$ dimension output, aka: a single number.

This would be written in pyTorch as: `einsum('i,i->', a, b)`.

Specific Numbers: If we let $\mathbf{a} = [1, 2, 3]$, and $\mathbf{b} = [-1, 0, -3]$, then:

$$\mathbf{a}_i \mathbf{b}_i = \sum_i \mathbf{a}_i \mathbf{b}_i = (1 \cdot -1) + (2 \cdot 0) + (3 \cdot -3) = -10$$

3.3 Example 2: Matrix Multiplication

Consider input matrices $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{m \times q}$. If we want to perform matrix multiplication, then, using symbolic Einstein notation, write:

$$\mathbf{A}_{i,j} \mathbf{B}_{j,k} = \mathbf{C}_{i,k}$$

[1] We usually keep the ordering of the indices the same, although it is not necessary

Since j is repeated, we must sum across that dimension. Since i, k each appear in only one of the input matrices, they are passed through to be dimensions in the output. Thus, we would expect $\mathbf{C} \in \mathbb{R}^{n \times q}$, since the length of $\mathbf{A}$ along i is n, and the length of $\mathbf{B}$ along k is q. Indeed this corresponds to the dimensions we would normally expect from matrix multiplication.

To calculate this, we would need to perform the following: *for every i, k in $\mathbf{C}$, we must perform the sum along dimension j.* Thus:

$$\mathbf{C}_{i,k} = \sum_j \mathbf{A}_{i,j} \cdot \mathbf{B}_{j,k}$$

This would be written in pyTorch as: `einsum('ij,jk->ik', A, B)`

Specific Numbers: If we let $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$ then we would calculate:

$$\mathbf{C}_{1,1} = \sum_j \mathbf{A}_{1,j} \cdot \mathbf{B}_{j,1} = (1 \cdot 1) + (2 \cdot 5) + (3 \cdot 9) = 38$$

$$\mathbf{C}_{1,2} = \sum_j \mathbf{A}_{1,j} \cdot \mathbf{B}_{j,2} = (1 \cdot 2) + (2 \cdot 6) + (3 \cdot 10) = 44$$

...

$$\mathbf{C}_{2,4} = \sum_j \mathbf{A}_{2,j} \cdot \mathbf{B}_{j,4} = (4 \cdot 4) + (5 \cdot 8) + (6 \cdot 12) = 128$$

Overall, $\mathbf{C} = \begin{bmatrix} 38 & 44 & 50 & 56 \\ 83 & 98 & 113 & 128 \end{bmatrix}$

We could also switch the order of the output indices to become: `einsum('ij,jk->ki', A, B)`. This would yield:

$$\mathbf{C} = \begin{bmatrix} 38 & 83 \\ 44 & 98 \\ 50 & 113 \\ 56 & 128 \end{bmatrix}$$

3.4 Other Examples

Matrix Dot Product: Consider input matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times m}$. Then, the matrix dot product can be calculated as:

$$\underbrace{\mathbf{A}_{i,j}\mathbf{B}_{i,j}}_{\text{Einstein}} = \sum_{i,j} \mathbf{A}_{i,j}\mathbf{B}_{i,j} = \mathbf{A} \cdot \mathbf{B}$$

This would be written in pyTorch as: `einsum('ij,ij->', A, B)`.

Vector Outer Product: Consider input vectors $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$. To compute the outer product $\mathbf{C} \in \mathbb{R}^{n \times m}$:

$$\underbrace{\mathbf{a}_i \mathbf{b}_j}_{\text{Einstein}} \Rightarrow \mathbf{C}_{i,j} = \mathbf{a}_i \cdot \mathbf{b}_j$$

This is written as: `einsum('i,j->ij', a, b)`.

Batched Dot Product Given two 3D tensors, $\mathbf{A}, \mathbf{B}$, you can compute the batched dot product between each matrix, with: `einsum('ijk,ijk->i', A, B)`. Thus, you end up with a vector of length corresponding to the ith dimension (number of batches in batch first representations) with all the dot products between the matrices.

Using Arbitrary Dimensions / Batched Matrix Multiplication: If you have two tensors of unknown dimension, but you want to compute the matrix multiplication along the last two dimensions, you can use the ellipses notation common to numpy and pyTorch. This would be written as: `einsum('...ik,...kj->...ij', A, B)`.

Dimensions Are Out Of Order: If you have $\mathbf{A} \in \mathbb{R}^{1 \times 3}$ and $\mathbf{B} \in \mathbb{R}^{2 \times 3}$, you can calculate the matrix multiplication $\mathbf{B}^T \mathbf{A} \in \mathbb{R}^{2 \times 1}$ by: `einsum('ij,kj->ki', B, A)`.

4 When Should I Use It?

As we showed in the previous sections, einsum can be used for an incredible number of operations, including common functions such as matrix product, inner products, and dot products. However, it is **not** recommended to use einsum for everything. The flexibility of the operation comes with a higher compute cost. Library implementations of einsum compute intermediate steps that can be made more efficient if you have prior knowledge of the specific operation taking place. Essentially, if there exists a specific library function for the operation (like matmul or dot) use that instead. Einsum is great for when there is no other convenient library function available.