Euclidean Distance Matrix Trick

Samuel Albanie
Visual Geometry Group
University of Oxford
albanie@robots.ox.ac.uk
June, 2019

Abstract

This is a short note discussing the cost of computing Euclidean Distance Matrices.

1 Computing Euclidean Distance Matrices

Suppose we have a collection of vectors $\{\mathbf{x}_i \in \mathbb{R}^d : i \in \{1, \dots, n\}\}$ and we want to compute the $n \times n$ matrix, D, of all pairwise distances between them. We first consider the case where each element in the matrix represents the squared Euclidean distance (see Sec. 3 for the non-square case)[1], a calculation that frequently arises in machine learning and computer vision. The distance matrix is defined as follows:

$$D_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||_2^2 \tag{1}$$

or equivalently,

$$D_{ij} = (\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) = ||\mathbf{x}_i||_2^2 - 2\mathbf{x}_i^T\mathbf{x}_j + ||\mathbf{x}_j||_2^2 \tag{2}$$

There is a popular "trick" for computing Euclidean Distance Matrices (although it's perhaps more of an observation than a trick). The observation is that it is generally preferable to compute the second expression, rather than the first[2].

Writing $X \in \mathbb{R}^{d \times n}$ for the matrix formed by stacking the collection of vectors as columns, we can compute Eqn. 1 by creating two views of the matrix with shapes of $d \times n \times 1$ and $d \times 1 \times n$ respectively. In libraries such as `numpy`,`PyTorch`,`Tensorflow` etc. these operations are essentially free because they simply modify the meta-data associated with the matrix, rather than the underlying elements in memory. We then compute the difference between these reshaped matrices, square all resulting elements and sum along the zeroth dimension to produce D, as shown in Algorithm 1.

Algorithm 1: Naive computation of Euclidean distance matrix

	Storage	**MACs**
Input: $X \in \mathbb{R}^{d \times n}$	$d \times n$	-
$A \leftarrow \texttt{reshape}(X, (d, n, 1))$	-	-
$B \leftarrow \texttt{reshape}(X, (d, 1, n))$	-	-
$C \leftarrow A - B \in \mathbb{R}^{d \times n \times n}$	$d \times n \times n$	$d \times n \times n$
$D \leftarrow \mathbf{1}_d^T C$	$n \times n$	$d \times n \times n$
Totals	$n^2(d+1) + nd$	$2dn^2$

[1] The term *Euclidean Distance Matrix* typically refers to the squared, rather than non-squared distances [1].
[2] It's mentioned, for example, in the metric learning literature, e.g. [2].

Note that the computation of C assumes that the matrix library will perform broadcasting, which implicitly forms A and B as $d \times n \times n$ matrices by repeating their elements along the singleton dimension, without allocating the memory for these expansions. However, memory *is* required for storing the result of the operation. To estimate the computational cost, we count the number of MACs (Multiply-Accumulate ops) required for each operation.

The alternative approach, which corresponds to computing the the expanded formula given in Eqn. 2, is given in Algorithm 2.

Algorithm 2: Expanded computation of Euclidean distance matrix

	Storage	**MACs**
Input: $X \in \mathbb{R}^{d \times n}$	$d \times n$	-
$G \leftarrow X^T X \in \mathbb{R}^{n \times n}$	$n \times n$	$d \times n \times n$
$D \leftarrow \text{diag}[G] + \text{diag}[G]^T - 2G$	$n \times n$	$2 \times n \times n$
Totals	$2n^2 + dn$	$n^2(d+2)$

The matrix G here is often referred to as the *Gram* matrix, and the diag$[G]$ operation simply selects the diagonal elements from G and stores them into an $n \times 1$ vector (we again use broadcasting in the final line to sum the vectors into a square).

Observations: There is an important different in the storage costs: in Algorithm 1, the $n^2 d$ term required to store the tensor representing all pairwise distance vectors can often become prohibitively large in practice. Moreover, for almost all values of n and d the first algorithm requires approximately twice as many MACs.

2 Non-Squared Euclidean Distance Matrices

Regardless of which algorithm is used, simply square root the matrix entries element-wise ($n \times n$ operations).

3 Numerical Stability

There are a couple of slight numerical challenges which it pays to be aware of when computing these matrices.

Negative Distances: It is possible (when using Algorithm 2) to get negative values in the matrix due to a lack of floating point precision. This is readily fixed by simply clamping each value to be non-negative: $D_{ij} \leftarrow \max(0, D_{ij})$.

Backpropagation Challenges: When the non-squared (rather than squared) matrix of distances is desired and the objective is not only to compute the distance matrix, but also to differentiate through its elements (e.g. as done in backpropagation), some care is required (particularly when using an Automatic Differentiation toolbox).

The issue stems from the element-wise square rooting operation. Since $d(\sqrt{x}) = \frac{1}{2\sqrt{x}} dx$, any zero values in the distance matrix will produce infinite gradients. This is encountered, for example, when implementing a contrastive loss [3] (see [4] for details). It can be addressed by adding a small value to matrix values immediately prior to performing the square root. The choice of value here is somewhat arbitrary [5], but depends on the floating point being used e.g. 10^{-16} for double precision.

References

[1] Jon Dattorro. *Convex optimization & Euclidean distance geometry*. Lulu. com, 2010.

[2] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4004–4012, 2016.

[3] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[4] Implementation of contrastive loss. https://leimao.github.io/article/Siamese-Network-MNIST. Accessed: 2019-06.

[5] Tom Murphy VII. What, if anything, is epsilon? In *Conference in Celebration of Harry Q. Bovik's 26th Birthday*, 2014.