I Symbolic matrices

We can represent tensors as:

(a) Dense matrix
Coefficients only

(b) Sparse matrix
Coordinates + coeffs

(c) Symbolic matrix
Formula + data

II An extension for PyTorch, NumPy, etc.

Our KeOps library comes with all the perks of a deep learning library:

• A transparent array-like interface.
• Full support for automatic differentiation.
• A comprehensive collection of tutorials, available online.

We support arbitrary formulas and variables with a wide range of Reductions: sum, log-sum-exp, K-min, matrix-vector product, etc.
Operations: +, x, sort, exp, neural networks, etc.
Advanced schemes: batch processing, block sparsity, etc.

Here is how to perform a fast nearest neighbor search:

1. Create large point clouds using standard PyTorch syntax:

   ```python
   import torch
   x, y = torch.randn(N, D), torch.randn(M, D) # (N, 10D) array
   D = ((x - y)**2).sum(dim=2) # (N, M) symbolic distances
   ```

2. Turn dense arrays into symbolic matrices:

   ```python
   from pykeops.torch import LazyTensor
   K = LazyTensor(D) # (N, M) symbolic K
   ```

3. Create a large symbolic matrix of squared distances:

   ```python
   D = ((x - x)**2).sum(dim=2) # (M, M) symbolic
   ```

4. Use an `.argmin()` reduction to perform a nearest neighbor query:

   ```python
   indices = D.argmin(dim=1) # => standard torch tensor
   ```

The line above is just as fast as the bruteforce (“Flat”) CUDA scheme of the FAISS library... And can be used with any metric!

III Applications

Symbolic matrices are to geometric ML what sparse matrices are to graph processing.

For geometric applications in dimension 1 to 100, KeOps symbolic tensors:

• Have a negligible memory footprint.
• Provide a sizeable speed-up for geometric computations.
• Always rely on bruteforce computations.
• Are less interesting when the formula \( F(x,y) \) is too large.

Our top priority for early 2021 is to mitigate these weaknesses: we will add support for Tensor cores and standard approximation strategies.

Overall, we believe that KeOps will stimulate research on:

• Clustering algorithms and UMAP-like methods,
• Kernel methods and Gaussian processes,
• Optimal transport theory,
• Geometric deep learning and shape analysis,
• And, even possibly, natural language processing!

We’ll be happy to discuss these questions with you!

You’re welcome to check our paper, visit: www.kernel-operations.io and the in-depth tutorial “Geometric data analysis, beyond convolutions”:

www.jeanfeydy.com/geometric_data_analysis.pdf