Typing linear algebra: A biproduct-oriented approach (Macedo and Oliveira, [2013])

Overview.

In this paper, the authors describe a categorical framework for reasoning about linear algebra operations.

Throughout the paper, the authors draw several connections between their construction and the "Algebra of Programming" [1]. For example, they describe the duality between the biproduct projections and injections in the terms of relational algebra.

The authors begin with a standard categorical formulation of matrices as morphisms in an abelian category where matrix dimensions are objects. They then explore how the biproduct projections and injections in this category define block matrices and the standard identities of matrix algebra, and build on this to derive a variety of algorithms for matrix multiplication.

The authors also extend their framework to explore how biproduct laws can define block-matrix formulations of various linear algebra operations. For example, they show how the rules for manipulating biproduct projections and injections defines the set of operations that we are "allowed" to do when performing Gauss-Jordan elimination. They also define a blocked Kronecker product based on their biproduct construction and the basic rules of monoidal categories.

The authors also present a novel formulation of vectorization algebra in terms of their categorical constructions. They build on this formulation to show how their construction significantly simplifies the calculation of the commutation matrix for transposing vectorized matrices.

Comments.

The authors' construction builds on a minimal set of assumptions about the category of matrices. They avoid any reference to the structure of the matrices or the spaces that they map between, which makes their proofs clean and index-free. Furthermore, the algorithms they derive make minimal assumptions about their inputs, and are therefore highly extensible. For example, the authors' derivation of Gauss-Jordan Elimination makes no reference to the field over which the input matrix is defined.

The authors also discuss how typing matrices and linear algebra constructions may simplify some programs. By expressing matrix dimensions as types, we certainly reduce the risk of bugs in situations where matrix sizes can be challenging to reason about, such as sequence-to-sequence recurrent neural network implementations. Moreover, we may be able to take this approach farther by leveraging dependent types to assign more complex types to matrices, such as "orthogonal", "unitary" or "identity". This would permit an extremely expressive style for programming matrix algorithms.

However, core linear algebra constructions need to be extremely fast, and most successful matrix libraries deal with a range of low-level considerations like memory access and management. In order for higher-level programs to take advantage of this speed and maintain their elegance and interpretability, most numerical computing systems abstract linear algebra routines with a high-level interface. To take advantage of the authors' ideas, we would need to change the nature of this high-level to low-level "handoff" and express many of the optimizations that are currently handled by low-level systems at higher levels of abstraction. The authors' treatment of vectorization is a step in this direction.

REFERENCES

[1] Bird, R. and De Moor, O. (1996). The algebra of programming.

[2] Macedo, H. D. and Oliveira, J. N. (2013). Typing linear algebra: A biproduct-oriented approach. *arXiv preprint arXiv:1312.4818*.