Dataflow Matrix Machines: a Collaborative Research Agenda

Michael A. Bukatin

December 27, 2019

Dataflow matrix machines form a quintessential interdisciplinary field. They emerged as a class of neural machines expressive enough to also serve as a viable programming framework. Their historical roots are in the synthesis of domains for denotational semantics and vector spaces. There are deep connections between vector semantics of programming languages and fuzzy and multivalued logic of partial inconsistency. The dynamical systems based on dataflow matrix machines exhibit a variety of interesting emerging properties.

As a programming framework, dataflow matrix machines have affinity with synchronous versions of dataflow and functional reactive programming. They generalize digital audio synthesis based on composition of unit generators (transformers of streams of numbers), thus providing potential to generalize the style of programming via composition of unit generators to visual animations, virtual reality, and eventually to general-purpose programming.

A class of spaces of V-values (flexible tensors based on tree-shaped indices) is extremely convenient for a variety of purposes. V-values allow to bring conventional data structures into the neural context, are convenient for hierarchies, are used to allow variadic activation functions, and have good potential for use in creating and training flexible neural interfaces between pre-existing software systems.

When considered as neural machines, dataflow matrix machines are remarkable for their strong self-referential facilities, allowing a neural network of this class to analyze and modify its current configuration on the fly. They form a natural framework for modular neural networks. This suggest a strong potential for their use in learning to learn, and also in neuroevolutionary methods.

Dataflow matrix machines were discovered and studied by a series of small-scale academic research collaborations. To unlock their full interdisciplinary potential, it would be necessary to generate a wider interest in this class of programmable neural machines.

In this note, I provide a bit more details and key references for some of the promising interdisciplinary research directions I see at the moment. I hope readers from various fields would find this of interest.

1 Background: how they work

The essence of neural model of computations is that linear and non-linear computations are interleaved. Hence, the natural degree of generality for neuromorphic computations is to work not with streams of numbers, but with arbitrary streams supporting the notion of linear combination of several streams (**linear streams**).

Dataflow matrix machines (DMMs) form a novel class of neural machines, which work with wide variety of **linear streams** instead of streams of numbers. The neurons have arbitrary arity (arity of a neuron can be fixed or variable). Of particular note are self-referential facilities: ability to change weights, topology, and the size of the active part of the network dynamically, on the fly, and the reflection capability (the ability of the network to analyze its current configuration).

There are various kinds of linear streams. They include streams of numbers, sparse vectors and sparse tensors (both of finite and infinite dimension), streams of functions and distributions. We found streams of V-values (**flexible tensors** based on tree-shaped indices) to be of particular use.

A single dataflow matrix machine can process a large variety of different kinds of linear streams, or it can be based on a single kind of linear streams, sufficiently expressive for a given class of situations.

This allows us to obtain neural machines which combine **general-purpose programming powers of stream-oriented architectures** such as traditional dataflow programming and more novel functional reactive programming with **good machine learning properties of conventional neural networks.**

Dataflow Matrix Machines resources:

Reference paper: `https://arxiv.org/abs/1712.07447`

Reference slide deck: `https://researcher.watson.ibm.com/researcher/files/us-lmandel/aisys18-bukatin.pdf`

GitHub Pages: `https://anhinga.github.io`

Open source implementation (Clojure): `https://github.com/jsa-aerial/DMM`

2 Conventional programming and program synthesis

The dimension of the network and the dimension of data are decoupled, so compact neural machines for solving conventional programming problems are available. For example, by considering streams of maps from words to numbers, one can build a dataflow matrix machine counting words in a given text which uses only a few neurons (Section 3 of `https://arxiv.org/abs/1606.09470`). Similarly, by considering streams of V-values (flexible tensors based on tree-shaped indices) and embedding of lists into trees, one can build a similarly compact dataflow matrix machine accumulating a list of asynchronous incoming events (e.g. mouse clicks, see Section 6.3 of the DMM reference paper, `https://arxiv.org/abs/1712.07447`).

The task of synthesis of dataflow matrix machines should be more tractable than conventional program synthesis. When one works with DMMs, the task of *learning program sketches* is reformulated as *neural architecture search*, and converting a program sketch to a full program should be done by conventional methods of neural net training.

Dataflow matrix machines allow us to combine

- aspects of *program synthesis* setup
  (compact, human-readable programs);
- aspects of *program inference* setup
  (continuous models defined by matrices).

3 Self-modification, learning to learn, and neuroevolution

Using neural networks for metalearning is always non-trivial. In particular, dimension mismatch, namely the number of neuron outputs being much smaller than the number of network weights, means that a neural network can only modify itself in a highly constrained manner. Dataflow matrix machines address this problem and have **powerful and flexible self-modification facilities**.

Therefore, a dataflow matrix machine can be equipped with a variety of primitives which perform self-modifications, and it can fruitfully learn various linear combinations and compositions involving those primitives.

Self-modification facilities of dataflow matrix machines are not limited to the weight changes for the existing connections in the network. The available primitives allow to modify the network topology as well. For example, primitives allowing the network to control its own fractal-like growth by the means of cloning its own subnetworks are available.

Therefore, this is a very promising architecture not only for methods of learning to learn better in a traditional sense, but also for methods of learning to perform neural architecture search better.

A dataflow matrix machine can comfortably host an evolving population of other DMMs inside itself, so it is an excellent environment for neuroevolution experiments and, in particular, for the experiments aiming to learn to evolve better (or to evolve to evolve better).

In our software experiments, we used self-modification facilities to

- produce controlled wave patterns in the network matrix (see Appendix B.2 of our LearnAut 2017 paper, `https://arxiv.org/abs/1706.00648`);
- create randomly initialized self-referential DMMs which generated interesting emerging behaviors (see Section 1.2 of our 11-2018 technical report, `dmm-notes-2018.pdf`);
- edit a running network on the fly by sending it requests to edit itself (in particular, this enables **live-coding**, but this is also quite open-ended, since it enables a population of networks to tell each other to modify themselves; of course, the receiving network doesn't have to follow an incoming instruction to self-modify blindly, although in the most simple-minded case it would do so; see Section 1.1 of our 11-2018 technical report, `dmm-notes-2018.pdf`).

4 Dynamical systems based on DMMs and emerging properties

The dynamical systems based on dataflow matrix machines exhibit a variety of interesting emerging properties. We conducted two series of experimental studies which yielded interesting emerging properties. These experiments were implemented in Processing programming language.

The first series of experiments was conducted in August 2015 and involved continuous cellular automata (see Section 4 of the preprint introducing the class of neural machines which we later started to call DMMs, `https://arxiv.org/abs/1601.01050`). The following videos show some of the emerging patterns we observed:

- `https://youtu.be/KZHQxdZUlSU`
- `https://youtu.be/-pFil1_GEA4`

The second series of experiments was conducted in 2016-2018 and involved "pure lightweight dataflow machines" (see Section 1.2 of our 11-2018 technical report, `dmm-notes-2018.pdf`). We observed, for example, the following emerging patterns:

- `https://youtu.be/_mZVVU8x3bs` - emerging sleep-wake patterns
- `https://youtu.be/CKVwsQEMNjY` - emerging oscillations

This is a fertile setup to discover new emerging patterns of behavior, to study those patterns mathematically, and, perhaps, to eventually design novel emerging patterns of behavior.

5 Synchronous vs. async, and artificial vs. biological neural nets

Like all neural machines, DMMs tend to be synchronous, because it is much easier to combine several streams with coefficients in the synchronous model of computations. As such, they have affinity with synchronous dataflow and functional reactive languages and libraries, such as, for example, `Lucid Synchrone` programming language and `Yampa` (a Haskell library).

By default, DMMs tend to provide extreme sparseness in connectivity patterns ("sparseness in space"), but not sparseness in time.

On the other hand, biological networks are asynchronous, providing sparseness in time, just like general dataflow programming tends to be asynchronous, and general functional reactive programming, in particular, tends to provide sparseness in time.

At the same time, biological neural networks tend to be equipped with various synchronization mechanisms. E.g. on the lower level, mechanisms such as, for example, **leaky integrate-and-fire** enable combining asynchronous inputs with coefficients. On higher levels, there are various mechanisms synchronizing firing times of different neurons (see, for example, research by the group led by Nancy Kopell at Boston University starting from around 2005 for various mathematical models of those mechanisms).

Reconciling synchronous and asynchronous models of computation, and, in particular, borrowing from what we know about biological neural nets and bringing it into realm of artificial neural machines is an important direction. Just like all "sparseness in time", it is also quite relevant to the field of low energy neural computations, which keeps growing in importance.

6 Flexibility vs. parallelization and optimization

Extreme flexibility of DMMs is provided by the use of V-values (flexible tensors with tree-shaped indices, see Section 3 the DMM reference paper, `https://arxiv.org/abs/1712.07447`, for the theory of V-values), sparse connections, and the ability of the network to self-reconfigure on the fly.

Obviously, there is a lot of tension between that and the ability to provide an efficient implementation, and especially with the ability to provide parallelized, GPU-friendly, and batching-friendly implementation.

The task of doing so is not impossible, but can be quite involved (see, for example, TensorFlow `Fold`, a library for working with dynamic computation graphs). See Appendix F of `https://arxiv.org/abs/1610.00831` for further discussion.

We started the initial design work towards reconciling tree-shaped tensor indices and GPUs/TPUs in `https://github.com/anhinga/2019-design-notes` repository. It is also promising to consider more flexible

parallel architectures than GPUs, such as, for example, architectures based on FPGAs, and also commercial flexible alternatives to GPUs, which are under development at a number of organizations.

This is an important and potentially very fruitful area of collaboration between specialists in parallel and efficient implementation of algorithms and people focusing on flexible neural architectures.

7 Links to fuzzy and multivalued logic of partial inconsistency

The historical roots of dataflow matrix machines are in the synthesis of domains for denotational semantics and vector spaces. It was that synthesis, which informed us that it was likely that programming with linear streams could be made powerful enough to be used for general-purpose programming.

The key element of that synthesis is the ability to handle partial contradictions (that is, to handle overdefined elements in addition to partially defined elements). For interval arithmetic, this corresponds to introduction of pseudosegments $[a, b]$ with the contradictory property that $b < a$ (the first known discovery of those overdefined interval numbers is by Mieczyslaw Warmus in his "Calculus of Approximations", 1956). For probability theory, this corresponds to allowing negative values of probabilities, which originally comes from physics (e.g. as quasiprobabilities by Eugene Wigner in 1932, and then as quasiprobabilities for phase-space formulation of quantum mechanics independently developed by José Moyal and Hilbrand Groenewold in 1940-es).

For the detailed overview of the material linking vector semantics with partial inconsistency, see Section 4 (and, in particular, Sections 4.2-4.4, and Sections 4.7-4.12) of our GCAI 2015 paper, "Linear Models of Computation and Program Learning", `https://easychair.org/publications/paper/Q4lW`. Because in addition to Lawvere duality between fuzzy orders and quasi-metrics, a similar duality exists between multivalued equalities and partial metrics (`https://www.cs.brandeis.edu/~bukatin/distances_and_equalities.html`), here we have both partial metrics and fuzzy multivalued predicates taking values in Warmus' partially inconsistent interval numbers.

There are interesting possible connections to be explored between this material and probabilistic logic networks by Goertzel et al. (2008).

8 Real-time functional programming: visual animation and virtual reality via composition of unit generators

Dataflow matrix machines generalize digital audio synthesis based on composition of unit generators (transformers of streams of numbers). Real-time digital audio synthesis and real-time generation of computer animation and virtual reality form very natural classes of soft real-time programming.

Moving from streams of numbers to more general linear streams, such as streams of V-values, should provide sufficient expressive power to program 2D and 3D visual animation based on composition of unit generators understood as transformers of linear streams. We explored this approach in various pre-DMM prototypes, and later in DMM systems, both for fixed computational graphs and for computational graphs dynamically changing on the fly. For our initial pre-DMM work in this direction see, for example, Sections 3-5 of `https://arxiv.org/abs/1601.00713` and associated videos:

- `https://youtu.be/fEWcg_A5UZc` - fixed dataflow graph
- `https://youtu.be/gL2L7otx-qc` - dataflow graph changing on the fly

Note that the old-fashioned analog video synthesizers also work in the style of composition of unit generators. Modern computer graphics tends to be imperative, oriented towards specific data flows implemented inside GPUs, and to require the software practitioners to focus on manual optimizations. The promise of doing animations via functional reactive programming is to focus again on semantically meaningful data flows, and to leave the hardware-oriented optimization to the underlying system.

I hope that collaboration between people specializing in visual effects and computer animation, people who understand how to compile flexible data flows and dynamic computation graphs to GPUs, and people who focus on DMMs will make this promise a reality.