Towards Practical Use of Dataflow Matrix Machines

Michael A. Bukatin

March 17, 2021 - December 26, 2024

Dataflow matrix machines (DMMs) form a class of neural machines with very interesting theoretical properties. On the level of single neurons, they replace streams of numbers by arbitrary streams allowing linear combinations of several streams (**linear streams**). It turns out that this change and a few other modifications make the resulting formalism suitable for general-purpose stream-oriented programming with **continuously deformable programs**. At the same time, these neural machines are expressive enough to provide convenient and flexible facilities for **compositional metalearning**.

We created experimental research-grade open-source implementations of self-referential dataflow matrix machines in Processing (with traditional mutable matrices) and in Clojure (with immutable streams of tree-shaped "flexible tensors"), and conducted a number of open-source software experiments using these implementations.

It is time to create a more professional implementation of DMMs in one of the modern ultra-flexible frameworks for differentiable programming such as Julia Flux or JAX, and to start using dataflow matrix machines in machine learning applications. In particular, one should be able to fruitfully target applications in **program synthesis** and in **metalearning**.

1 Introduction

Many of us know the history of ReLU use in neural networks. In the year 2000, it became apparent that the use of ReLU induced semantically meaningful sparsity[1]. At that moment one should have been able to conclude that ReLU was a very promising activation function. However, the first papers showing that the use of ReLU actually improved performance on particular benchmarks had only appeared in 2009-2011[2], and only then widespread adoption of ReLU had actually started.

I believe that we currently have a somewhat similar situation with DMMs. Their theoretical properties are very attractive, but the hard work of demonstrating that DMMs can be used to improve performance on particular benchmarks has not been done yet.

Moreover, when DMMs were introduced in 2015-2017, the existing machine learning frameworks were still rather rigid, and implementing the most flexible varieties of DMMs would take a lot of labor in frameworks such as, for example, TensorFlow or PyTorch.

The situation is much more favorable now with the availability of modern ultra-flexible frameworks for differentiable programming such as Julia Flux or JAX, which seem to fit DMMs quite well.

1.1 Current Status

JAX and Zygote.jl (Julia Flux) are capable of taking gradients with respect to variables stored inside nested dictionaries[3].

First successful experiments in DMM training and in program synthesis/circuit synthesis/DMM synthesis via neural architecture search were performed in June 2022 using Zygote.jl. The synthesized DMMs had pretty impressive generalization properties.[4].

[1]Richard Hahnloser, Rahul Sarpeshkar, Misha Mahowald, Rodney Douglas, H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, **405**, pp. 947–951 (2000). The reasons for sparsity induction by ReLU are similar to the reasons for sparsity induction by L1 regularization, which not surprising given that L1 regularization adds the regularization term $\alpha \sum(\text{ReLU}(w_{ij}) + \text{ReLU}(-w_{ij}))$ to the loss function.

[2]Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, Yann LeCun. *What is the Best Multi-Stage Architecture for Object Recognition?*, ICCV'2009; Vinod Nair, Geoffrey Hinton, *Rectified Linear Units Improve Restricted Boltzmann Machines*, ICML'2010; Xavier Glorot, Antoine Bordes, Yoshua Bengio. *Deep Sparse Rectifier Neural Networks.* AISTATS'2011.

[3]See `https://github.com/anhinga/jax-pytree-example` and `https://github.com/anhinga/julia-flux-drafts/tree/main/arxiv-1606-09470-section3`

[4]See `history.md` and `JuliaCon2023-talk` at `https://github.com/anhinga/DMM-synthesis-lab-journal`

2 Choice of Applications

We maintain a list of open problems and promising directions of research related to DMMs[5] in order to facilitate potential interdisciplinary collaborations. I am including the first three sections from that list verbatim as an appendix.

Here I briefly mention the main directions.

2.1 Program Synthesis

The task of synthesizing a neural machine should be much simpler than a general program synthesis task (the progress in neural architecture search seems to be ahead of the progress in conventional program synthesis).

2.2 Metalearning

It seems that some kinds of metalearning in neural machines can benefit from increasing the dimension of network output (or network hidden state) relative to the dimension of the space of its weights. The recent paper by Kirsch and Schmidhuber[6] seems to provide an experimental confirmation of that conjecture. Using DMMs would allow to push further in that direction.

2.3 Other sections of “Collaborative Research Agenda”

Section 6 provides more detailed considerations of implementation issues. Section 9 outlines what can be done to further leverage the fact that DMMs can handle streams of probabilistic samples of arbitrary nature without embedding those samples into vector spaces.

Section 10 notes that standard digital audio synthesis via composition of unit generators is essentially a flavor of irregular hand-crafted and hand-tuned neural networks using somewhat more powerful neurons (such as e.g. `f(x,a,b) = sin(a*x+b)`) and suggests that DMMs should allow us to synthesize visual animations in the same style. So if one is looking for a more concrete application area for DMMs, neural machines generating visual animations might be a particularly attractive choice.

Section 11 starts to analyze deep connections which exist between DMMs and attention-based models including Transformers. Promising directions of research include using what we know about DMMs to shed some light on the remarkable properties of Transformers and exploring the ways to incorporate key elements from Transformer architecture into a more flexible DMM setup. In particular, we are trying to see if it is possible to obtain interesting compact and low training cost models by incorporating attention-inspired and Transformer-inspired motives into DMMs.

A The first three sections of “Collaborative Research Agenda”

A.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.

[5]Michael Bukatin. *Dataflow Matrix Machines: a Collaborative Research Agenda*, September 2022. `https://anhinga.github.io/brandeis-mirror/dmm-collaborative-research-agenda.pdf`

[6]Louis Kirsch, Jürgen Schmidhuber, *Meta Learning Backpropagation And Improving It*, December 2020. `https://arxiv.org/abs/2012.14905`

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.**

There are deep connections between DMMs and attention-based models including Transformers. Each input of a neuron computes a linear combination of linear streams (which tend to be high-dimensional or infinite dimensional entities), so each input of each neuron performs a (generalized) attention operation. Transformer-like rewrites of DMM attention operations in terms of matrix multiplication are also available in many situations.

Dataflow Matrix Machines resources:
Reference paper: `https://arxiv.org/abs/1712.07447`
Reference slide deck: `https://github.com/jsa-aerial/DMM/blob/master/doc/IBM-AI-Systems-Day-2018/aisys18-bukatin.pdf`
GitHub Pages: `https://anhinga.github.io`
Open source implementation (Clojure): `https://github.com/jsa-aerial/DMM`

A.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`).

For more examples of DMMs as programs, see *Map of DMM-related programming examples and techniques*:
`https://github.com/anhinga/2020-notes/tree/master/programming-overview`

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).

First successful experiments in program synthesis/circuit synthesis/DMM synthesis via neural architecture search were performed in June 2022[7].

A.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.

[7]See `history.md` at `https://github.com/anhinga/DMM-synthesis-lab-journal`

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`).