Synergy between AI-generating algorithms and dataflow matrix machines

Michael A. Bukatin

March 22, 2020

AI-generating algorithms is an alternate paradigm for producing general artificial intelligence introduced by Jeff Clune in 2019. Its idea is to create an algorithm which itself automatically learns how to produce general AI. The approach is structured into three pillars: 1) meta-learning AI architectures, 2) meta-learning the learning algorithms, and 3) generating effective learning environments.

Dataflow matrix machines form a novel class of programmable neural machines bridging the gap between programs and recurrent neural networks. This class of neural machines seems to be suitable for general purpose programming, allows to conveniently express powerful self-modification facilities, and seems to be naturally tailored for the tasks of synthesizing modular neural architectures.

In this essay, I argue that there is deep synergy between AI-generating algorithms (AI-GAs) and dataflow matrix machines (DMMs). The particular focus of the present essay is an argument that DMMs are an extremely natural fit for the first two pillars of AI-GAs.

I also briefly overview the current state of DMM research, and consider further possibilities of potential fruitful interaction between AI-GAs and DMMs.

1 Introduction

The May 2019 essay by Jeff Clune[1] formulates a paradigm for automatically learning to produce general AI. The first two pillars of this paradigm, meta-learning efficient architectures and metalearning the learning algorithms, go back a long way[2]. The punchline of the AI-GAs essay is that these two pillars together with the novel third pillar of generating effective learning environments and training data are likely to be able to sufficiently accelerate each other to reach the level of human-equivalence or more given sufficient computational resources[3].

An important factor here would be our ability to tightly integrate work done under these three pillars. It would be much easier to make progress along these lines, if there is a unified platform suitable for this work, whereas a disconnect between algorithms and programs on one side and neural networks and their architectures on the other side would be a factor inhibiting progress of AI-GAs.

[1] Jeff Clune, *AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence*, `https://arxiv.org/abs/1905.10985`

[2] See e.g. Yann LeCun, John Denker, Sara Solla, "Optimal brain damage", in *Advances in neural information processing systems*, pp. 598-605, 1990 and Neil Cotter, Peter Conwell, "Fixed-weight networks can learn", in *1990 IJCNN International Joint Conference on Neural Networks*, pp. 553-559, IEEE, 1990.

[3] The question of how much computational resources is needed is quite non-trivial, and I further remark on it in Section 5.3.

In particular, it seems that the problem of program synthesis for the traditional programming platforms used by software engineers in their daily professional activities is much more difficult than the problem of synthesis of neural nets. Hence if neural nets can't provide the required programming capabilities in the pragmatic day-to-day software engineering sense (rather than much weaker theoretical sense of formal Turing completeness), this will remain a factor slowing AI-GAs progress given that it is necessary for AI-GAs to synthesize computer code.

I am going to make the case that traditional neural machines such as RNNs are not adequate in this sense, whereas DMMs seem to be reasonably adequate.

2 Problems with traditional neural nets as a programming platform

The current state of unification of programs and neural networks is not quite satisfactory. While the results establishing Turing-completeness of RNNs in the presence of unlimited memory go back to at least 1987[4], these results tends to rely on real numbers of unlimited precision and on using a binary expansion of a real number as a Turing machine tape. The consequence of such an approach is extreme sensitivity to small levels of noise, as one needs to perform the calculations to great precision, and even small noise destroys data encoded further in the binary expansion of the real number in question. Therefore, good learnability properties of neural networks are destroyed under this approach (and certainly the ability to use reduced precision computations and TPUs is utterly destroyed).

Generally, there is a boundary between pragmatically usable programming platforms and esoteric programming languages ("Turing tarpits"). Bare-bones RNNs are probably closer to the esoteric side of this divide, because of their lack of structurization capabilities and the need for imprecise learned embedding of discrete data structures into vector spaces[5].

Self-modification is also difficult. First studies of self-modifying neural networks go back to at least 1993[6]. However, there is the dimension mismatch: the number of network outputs is much smaller than the number of network weights. This dimension mismatch forces people to either do self-modification in a heavily constrained manner (e.g. limiting themselves to a low-dimensional subset in the space of possible network matrices), or to use "address multiplexing", which again creates systems highly sensitive to noise.

Dataflow matrix machines are designed to address all these problems of "RNNs as programs" and to provide us with a class of neural machines which is expressive enough to constitute a viable programming platform, to host structured information without distorting it by embeddings, to have rich and convenient self-modification facilities, and to be able to encapsulate any algorithms within neurons, as long as those algorithms agree to interface via streams of data for which one can *combine several streams with coefficients*.

3 DMM architecture

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**).[7]

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).

---

[4] Jordan Pollack. *On connectionist models of natural language processing.* PhD thesis, University of Illinois at Urbana-Champaign, 1987. Chapter 4 is available at `http://www.demo.cs.brandeis.edu/papers/neuring.pdf` (Chapter 4 contains the Turing-completeness proof); see also Hava Siegelmann and Eduardo Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50:132-150, 1995.

[5] Andrej Karpathy remarks in `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`: "it is known that RNNs are Turing-Complete in the sense that they can [...] simulate arbitrary programs (with proper weights). But similar to universal approximation theorems for neural nets you shouldn't read too much into this. In fact, forget I said anything.";
See also Edward Grefenstette, *Limitations of RNNs: a computational perspective*, NAMPI 2016 workshop at NIPS.

[6] Jürgen Schmidhuber, *A 'Self-Referential' Weight Matrix*, In: Gielen, S., Kappen, B., ICANN '93: Proceedings of the International Conference on Artificial Neural Networks, Springer, 1993, pp. 446-450.

[7] This section repeats Section 1 of Michael Bukatin, *Dataflow Matrix Machines: a Collaborative Research Agenda*, December 2019, `https://www.cs.brandeis.edu/~bukatin/dmm-collaborative-research-agenda.pdf`

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`

4 Future applications of the DMM architecture

Here we specifically focus on synthesis of algorithms and programs, and on learning to learn[8].

Note that in the world of DMMs there is no strict boundary between an architecture of the network and the algorithm it implements. Conceptually, we consider a countable address space and a countable-sized network, with only a finite number of weights being non-zero at any given moment of time and the corresponding finite part of the network being allocated in the computer memory and active at that moment of time. The dynamically changing network architecture corresponds to the *sparsity structure* of the network, i.e. to knowing which weights must be zero and which weights are allowed to be non-zero at the moment; this also corresponds to the notion of *program sketch* in the world of more traditional programming.

So, in some sense, the total of the first two AI-GAs pillars is structured a bit differently. Instead of distinguishing between learning the architectures and learning the learning algorithms, we distinguish between learning the architecture and learning the weights of a network capable of modifying its own weights and architecture. The primitives inside neurons can be arbitrary complex (a subnetwork can even be used inside a neuron), so supplying a good library of primitives is an important part of learning the architectures.

4.1 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 [9]. 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, for example, mouse clicks[10].

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).

[8] The two subsections below repeat Sections 2-3 of Michael Bukatin, *Dataflow Matrix Machines: a Collaborative Research Agenda*, December 2019, `https://www.cs.brandeis.edu/~bukatin/dmm-collaborative-research-agenda.pdf`

[9] Section 3 of Michael Bukatin, Steve Matthews, Andrey Radul, *Programming Patterns in Dataflow Matrix Machines and Generalized Recurrent Neural Nets*, June 2016. `https://arxiv.org/abs/1606.09470`

[10] Section 6.3 of Michael Bukatin, Jon Anthony, *Dataflow Matrix Machines and V-values: a Bridge between Programs and Neural Nets*, `https://arxiv.org/abs/1712.07447`, in Gyuris, B. et al. (eds.), *K + K = 120: Papers dedicated to László Kálmán and András Kornai on the occasion of their 60th birthdays*, Research Institute for Linguistics, Hungarian Academy of Sciences, 2017

4.2 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**[11].

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[12];
- 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)[13].

5 Further thoughts on interplay between AI-GAs and DMMs

5.1 Leveraging structuring capabilities of DMMs

One remark in the AI-GAs essay is that "it is likely that an AI-GA would produce a complex machine whose inner workings we do not understand", which is an undesirable property. In this sense, the ability of DMMs to form compact and understandable neural machines with rich functionality is a big advantage, and so is their ability to form modular, hierarchical network structures. It should be much easier to achieve machines which we can understand along this path, compared to the usual mixture of very large traditional neural nets and large amounts of conventional computer code.

The same set of properties of DMMs makes them good construction material for artificial virtual worlds and artificial civilizations. Instead of having people or artificial agents implement artificial worlds in conventional languages, and then define and use complicated APIs, we can simply build those worlds from DMMs. This way, we achieve uniformity of our material in all Three Pillars of AI-GAs, and we have a good degree of control of the extent to which the artificial worlds in question are alive and morphing vs. static and unchanging.

5.2 We expect DMMs to benefit from AI-GAs work

The prime vector of further DMM development is discovery of new DMM programming idioms and paradigms, and also discovery of new self-modification and machine learning patterns. Currently, we are at the stage when this work is being done manually, but eventually we expect this to become more and more automated.

[11] In DMMs, streams of network matrices can be handled by a single neuron, so a single neuron can emit the current network matrix on each step, while accepting updates to that matrix from other neurons. The network matrix does not have to be rewritten on each step, but can be changed incrementally.

[12] Section 1.2 of DMM technical report 11-2018. *Dataflow matrix machines: recent experiments and notes for next steps.* November 2018. `https://www.cs.brandeis.edu/~bukatin/dmm-notes-2018.pdf`

[13] see Section 1.1 of the same technical report

We do expect a typical AI-GAs advance to be applicable to DMMs in this sense, and to result in novel ways of composing and using DMMs. At some point, we expect the automated processes to match and exceed our own abilities of being creative with DMM design and use. But even before that, a variety of modest AI-GAs-related advances should produce interesting and unexpected DMM-related developments.

5.3 How much compute is needed for AI-GAs?

Jeff Clune writes: "The AI-GA philosophy is that via a compute-intensive, sample-inefficient outer loop optimization process we can produce learning agents that are extremely sample efficient and that generalize well." He also writes: "AI-GAs will require extraordinary amounts of computation by today's standards".

While I am certain that AI-GAs will be capable of creatively using any amounts of computation available to them (and will eventually be likely to generate new compute capability in the material world), I am less sure that AI-GAs will *necessarily* require extraordinary amounts of computation by today's standards.

The name of the game is to reduce the amount of compute from what it took the computer named Earth and its Biosphere to produce something like us, to the amount of compute feasible to us at some point. But whether we can only hope to cut the required computer time and memory to "extraordinary amounts of computation by today's standards", or whether we can cut it more drastically and, perhaps, even to something already computationally feasible today is difficult to predict.

The question here is: how huge must the outer loop computation be? The answer is not obvious at all, and very much depends on us being inventive in all Three Pillars.

We are aware of the modes which are intermediate between manual engineering and fully automated evolution. E.g. human-guided selection of animal breeds works quite rapidly, compared to unguided evolution by natural selection, and the associated computation is much more compact, compared to unguided evolution. Also, compositional pattern-producing networks require negligible computer resources to produce sophisticated patterns via *interactive evolution* (of course, interactive evolution also recruits computational power of participating humans).

There are also other ways to affect the speed of AI-generation in the mode of "weakly guided evolution", such as, for example, occasionally injecting a promising new type of "superneuron", or a promising connectivity pattern, which can then be automatically copied and modified.

So, this might be the only place where I don't quite agree with the intuition from the AI-GAs essay. I don't think there is a meaningful lower bound on the compute required along this path, and I don't think there is any ground for the statement that this program can't be successful with today's computer resources.

Of course, the more resources the better, and it is important to have an approach which scales up gracefully and can take advantage of as much compute as becomes available in the future.

5.4 Safety and ethical considerations

I have studied carefully the "Safety and ethical considerations" section of the AI-GAs essay, and I generally agree with what it says.

I gave a particular thought to the paragraph starting with sentence "It is fair to ask why should I write this paper if I think AI-GA research is more dangerous, as I am attempting to inform people about it potentially being a faster path to general AI and advocating that more people work on this path"[14]. I gave a lot of consideration on how this line of thought might be applicable to the present essay. The AI-GAs essay is published, and our own previous DMM-related work is published, but the present essay might constitute "the next step", and each time a potentially significant next step is done, the question of whether to make the results public, and if yes, in what form, arises again and needs to be revisited.

On one hand, one might ask whether the advance is "natural" and looks inevitable within a relatively short timeline. If we are talking about a very intricate and very non-obvious development, then one can imagine that there are futures where this development is not present or does not become common knowledge. On the other hand, if a development is very "natural" and feels "inevitable" and "already long overdue", then the question is really not whether this advance would be made public eventually (the terminology and personalities involved might differ, but the substance remains more or less invariant and will become public knowledge at some point). Rather, in that case, the question is what would be the timeline, and what will be the sequence of its introduction: who will work with these new techniques first, what will they use them for at first, what safeguards will they apply, etc.

[14] Section 4, page 20 of `https://arxiv.org/abs/1905.10985`

In the present case, I believe that both AI-GAs and DMMs, and also their interplay, are "natural, inevitable, and already long overdue". So, after some reflection, the approach I am taking is as follows. I am publishing this particular essay as a "GitHub preprint", and I also disseminate it in other relatively modest ways, but I don't want to advertise it too much at the moment.

I also try to emphasize approaches which bring human aesthetics into play early in the game. This is why many of our DMM-related programming exercises are related to interactive visual animations, and that's why I hope that visual and audio-visual art will continue to be a prominent motif during our further near-future explorations of AI-GAs, DMMs, and related topics, following the examples set by DeepDream, by compositional pattern-producing networks, etc. The hope here is "to make the future AGIs more analog and less digital *in spirit*, and more *like us* in this sense, in terms of their perception of the world, their aesthetics, and such".

So the approach I am taking at this point is trying to moderately influence the issues of who will work with these new techniques first and what will they use them for at first, while recognizing that one can only do so much in this sense, when we are taking about things which are as "natural, inevitable, and already long overdue" as the material in this essay seems to be.

A Appendix: current state of DMM research

Here I describe what has been done by me and my colleagues, and what I am working on actively at the moment. The key DMM resources (reference paper, reference slide deck, reference open-source implementation, and GitHub Pages site) are listed in Section 3. The aim of this Appendix is to informally complement those resources to assist readers who might be considering the possibility of actually using DMMs in their work.

The contrast between very flexible DMMs of maximal generality and subclasses of rigid DMMs will be made throughout this section. It is easier to work with rigid subclasses within the most popular machine learning frameworks existing today, such as PyTorch. However, looking further into the future, we should aim for working with very flexible DMMs of maximal generality. The machine learning frameworks which are actually tailored for this degree of flexibility, such as Julia Flux, started to emerge lately.

A.1 DMMs as a programming platform

We have explored a number of DMM programming techniques and examples in recent years[15].

More efforts should be applied towards creating new DMM programming techniques and examples.

We believe in pluralism of DMM programming styles. Traditional programming architectures support a large variety of programming styles: imperative, object-oriented, functional, logical, dataflow, and many others. We think that programming in the DMM paradigm will also support a diversity of programming styles. Some of those programming styles might be fairly conservative modifications of programming styles available in the realm of discrete programs (e.g. there is a lot of affinity to various stream-based architectures, such as traditional dataflow or more modern functional reactive programming). Other DMM programming styles to be discovered in the future might be entirely novel.

We also hope to get extra ideas for new DMM programming paradigms from implementing DMMs as embedded domain-specific languages (for example, our current reference open-source implementation is a DSL embedded into Clojure).

A.2 Research-grade implementations of DMMs as a programming platform

The reference research-grade[16] open-source implementation is in Clojure. It is a very flexible architecture based on immutable streams of flexible tensors with tree-shaped indices and unbounded network size (countable-sized address space). Because of immutability, the common data substructures are shared in memory.

We also have a more traditional open-source implementation of a subclass of rigid DMMs, *pure lightweight dataflow matrix machines*, based on mutable streams of network-sized matrices and networks of fixed finite size. This research-grade implementation is done in Processing[17]. This architecture is much more familiar to

[15] The maps of DMM programming techniques and examples currently present in DMM literature and in code are located at `https://github.com/anhinga/2020-notes/tree/master/programming-overview`

[16] "Research-grade" here means that we are trying to maintain sufficient software quality to enable complete understanding of this software and to facilitate its further use by people, and we try to support this software within reason and to provide timely response to issues, but we are not trying to create a widely used "professional" open-source implementation yet.

[17] See Section 1 of [DMM technical report 11-2018. *Dataflow matrix machines: recent experiments and notes for next steps.* November 2018. `https://www.cs.brandeis.edu/~bukatin/dmm-notes-2018.pdf`] for experiments with self-modifying DMMs in both of these implementations.

the neural net community, and is straightforward to translate to the commonly used ML frameworks.

We started exploratory efforts for research-grade open-source implementations in Python and in Julia. We would like to eventually integrate DMMs into PyTorch and into Julia Flux. In the context of this exploratory effort, we started to use multiple kinds of linear streams within one dataflow matrix machine[18], whereas our existing Clojure and Processing implementations are based on having a single sufficiently expressive kind of linear streams[19].

A.3 Design work for DMMs as a machine learning platform

While we have performed a number of experiments with self-modifying neural machines (DMMs), and while the class of DMMs includes known neural networks as subclasses, our group has only done preliminary design work for future machine learning experiments with DMMs.

Some of the dichotomies here are between gradient-based methods and gradient-free methods, and also between GPU acceleration and just using CPU cores.

For moderate scale experiments, one can simply use derivative-free methods and CPU cores. For example, as a derivative-free method, one can use the modern incarnation of evolution strategies, introduced by researchers from OpenAI[20] and further elucidated by researchers from Uber AI Labs[21].

We have also started to do exploratory work towards using an adaptive "population coordinate descent" derivative-free schema. The essence of that schema is to maintain an evolving population of directions which forms an overdefined coordinate system and to use an adaptive probability distribution/adaptive sampling schema to repeatedly sample a direction for the next step of coordinate descent[22].

However, looking forward, one really wants to have an option of using gradient-based methods and GPU/TPU acceleration. The most popular machine learning frameworks, such as PyTorch, are a nice fit for rigid subclasses of DMMs. However, they are oriented towards standard tensors (multidimensional arrays of a fixed number of dimensions and fixed sizes along each of those dimensions), and using them for the more flexible variety of DMMs based on flexible tensors with tree-shaped indices is non-trivial[23].

A better fit might be a machine learning framework, designed from start with the degree of flexibility we would like to have here. In particular, Julia Flux, "the ML library that doesn't make you tensor", might be a good fit for our use case[24]. Its incredibly flexible *Zygote* differentiable programming system is particularly impressive[25].

[18] Multiple kinds of linear streams within single DMM first appear in Michael Bukatin, Steve Matthews, Andrey Radul, *Dataflow matrix machines as programmable, dynamically expandable, self-referential generalized recurrent neural networks*, May 2016. `https://arxiv.org/abs/1605.05296`

[19] Streams of V-values are used in the Clojure implementation and streams of rectangular matrices of a fixed size are used in the Processing implementation.

[20] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, Ilya Sutskever, *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*, March 2017. `https://arxiv.org/abs/1703.03864`

[21] Xingwen Zhang, Jeff Clune, Kenneth Stanley, *On the Relationship Between the OpenAI Evolution Strategy and Stochastic Gradient Descent*, December 2017. `https://arxiv.org/abs/1712.06564`

[22] This work is still in its exploratory design stage: `https://github.com/anhinga/population-of-directions`

[23] A design sketch for one possible way of flattening and reshaping tree-shaped indices to fit the "fixed number of dimensions/fixed size" framework can be found here:
`https://github.com/anhinga/2019-design-notes/blob/master/automated-synthesis/flattening-of-v-values.md`

[24] `https://github.com/FluxML/Flux.jl`; the reference paper for Julia Flux is Michael Innes et al., *Fashionable Modelling with Flux*, November 2018, `https://arxiv.org/abs/1811.01457`

[25] Michael Innes, *Don't Unroll Adjoint: Differentiating SSA-Form Programs*, October 2018, `https://arxiv.org/abs/1810.07951` and Michael Innes et al., *A Differentiable Programming System to Bridge Machine Learning and Scientific Computing*, July 2019, `https://arxiv.org/abs/1907.07587`