Dataflow matrix machines: recent experiments and notes for next steps

DMM technical report 11-2018

November 2, 2018

Abstract

This document describes experiments with **self-referential dataflow matrix machines** performed by participants of DMM and Fluid projects in January-October 2018 and collects together various notes for use in possible next steps in DMM research.

Contents

Introduction

The intended audience of this document consists of people already familiar with dataflow matrix machines and either working on them actively or considering whether to do some work with dataflow matrix machines in the near future. The general audience is referred to the published literature and references therein[1].

Fragments of these notes might be reused in the subsequent publications by members of DMM and Fluid projects.

[1]Michael Bukatin, Jon Anthony, **Dataflow Matrix Machines and V-values: a Bridge between Programs and Neural Nets**, "K + K = 120" Festschrift. The paper: `https://arxiv.org/abs/1712.07447`. The talk: `https://youtu.be/X6GCohQ-LHM`. The slides: `https://researcher.watson.ibm.com/researcher/files/us-lmandel/aisys18-bukatin.pdf`.

The essence of neural model of computations is that linear and non-linear computations are interleaved. Therefore, 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) is a novel class of neural abstract machines, which work with arbitrary **linear streams** instead of streams of numbers. The neurons have arbitrary fixed or variable arity. Of particular note are self-referential facilities: ability to change weights, topology, and the size of the active part dynamically, on the fly, and the reflection capability (the ability of the network to analyze its current configuration).

The resulting computational architecture is highly expressive, but we have only started to explore various ways to use it. Our recent and next steps are directed towards teaching ourselves to harness the power of DMMs, while keeping the following two of the attractive longer-term goals in mind:

- **Learning to learn.**

  We expect that networks which modify themselves should be able to **learn to modify themselves better**.

- **Program synthesis.**

  DMMs are powerful enough to write programs.
  They provide a programming framework where one can
  deform programs in continuous manner.

  A program is determined by a matrix of numbers. Therefore, it is sufficient to synthesize a matrix of numbers to synthesize a program.

  DMMs combine

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

  We hope that this will make them a sweet spot for program synthesis.

Section 1 describes two recent series of experiments with self-referential dataflow matrix machines. The subsequent sections collect together various notes for possible use in the next steps of DMM research. These notes are loosely grouped into Sections 2 - 4.

1 Recent experiments with self-referential DMMs

1.1 Editing a running network on the fly

We conducted a series of experiments providing support for **livecoding** via editing a running network on the fly[2].

This series of experiments is done with DMMs based on V-values and variadic neurons implemented in Clojure. A dedicated update neuron is listening for update V-values on a `clojure.async` channel.

The further details are given for the latest experiments in this series (the April 2018 experiments). At the setup, the `:direct` output of the update neuron is connected with weight 1 to the `:delta` input of the `:self` neuron which accumulates the currently used network matrix.

The update neuron emits values it receives on its `clojure.async` channel. These values are emitted during the "up movement" (the empty map standing for zero V-value is emitted if nothing is received on the channel).

When the update neuron emits {`:direct` **X**} V-value, the V-value **X** is copied to the `:delta` input of the `:self` neuron during the next "down movement" and gets added to the network matrix during the subsequent "up movement".

These edits of the network matrix can connect other outputs of the update neuron to other neurons of the network, and then the update neuron can be used to send asynchronously received updates to those neurons as well. For example, in our April 2018 experiments we interactively connect the `:to-test-image` output of the update neuron to accumulator neuron `:test-image` (which is to be used to hold a V-value representing an image) by adding 1 to the element of the network matrix situated at row `[v-accum :test-image :delta]` and column `[v-network-update-monitor :network-interactive-updater :to-test-image]` .

After that we populate this image-holding neuron by sending the V-value {`:to-test-image` **V-value-representing-image**} to the update neuron.

1.2 Emerging bistability in randomly initialized DMMs

We conducted a series of experiments with randomly initialized DMMs[3].

This series of experiments is done with Lightweight Pure DMMs[4] based on streams of network-sized rectangular matrices implemented in Processing 2.

The first Lightweight Pure DMMs with random initialization of the output layer were created by GitHub user `nekel` in September 2016. These were the first self-referential DMMs which not only modified their own network matrix,

[2] `https://github.com/jsa-aerial/DMM/tree/master/examples/dmm/quil-controlled/interactive`

[3] `https://github.com/anhinga/fluid/tree/master/atparty-2018/game_of_afterlife`

[4] Lightweight Pure DMMs were introduced in Appendix D of Michael Bukatin, Steve Matthews, Andrey Radul, **Notes on Pure Dataflow Matrix Machines: Programming with Self-referential Matrix Transformations**, `https://arxiv.org/abs/1610.00831`. For relationship between Lightweight Pure DMMs and DMMs based on V-values and variadic neurons see Appendix F of that preprint.

but which also used the current values of their network matrix as non-trivial summands when forming their input layer.

For each network input, the corresponding row of the network matrix is used to form the value of that input during the "down movement". However, if the weight w connecting the given input and the output of `Self` is non-zero, the whole network matrix W also participates in the newly formed input value as $w \cdot W$ summand. The Lightweight Pure DMMs with random initialization of the output layers constitute the first example of this phenomenon.

In this series of experiments we sampled a seed for random number generator and recorded this seed, so that we can reproduce those runs which seem to be of interest. A significant fraction of those runs exhibited various emerging bistable dynamic patterns. We committed a number of those configurations to GitHub.

Our empirical observations seem to suggest that the tendency to have a significant fraction of random initializations for various network configurations to exhibit bistable patterns might be fairly universal in this context. At the same time, we have no theoretical understanding of the observed bistability.

The remark which might be useful for people who would like to do further work with the code we committed is that a `squeeze` function is applied to all output matrices[5], so all activation functions look like `squeeze` $\circ f$.

2 V-value transformers

DMMs based on V-values and variadic neurons can be considered as transformers of streams of V-values.

So, on one hand, machine learning problems in this context can be formulated as problems of synthesis of transformers of streams of V-values from a given set of primitives.

On the other hand, there is a task of creating a sufficiently rich library of built-in transformers of streams of V-values.

2.1 Software connectors

Since V-values are based on nested maps (and, therefore, are similar in spirit to JSON and such), thinking about various tasks of autogeneration of connectors between various pieces of software as tasks of autogeneration of transformers of V-values and streams of V-values is quite natural[6].

This might be one of the more straightforward roads to pragmatically useful program synthesis (instead of focusing on synthesizing small programs from scratch, one might focus on automating the practice of configuring software from a small number of large pre-existing software components)[7].

[5] Each matrix is divided by the maximal absolute value of all its elements.

[6] Cf. Section 5 of Michael Bukatin, Steve Matthews, **Linear Models of Computation and Program Learning**, GCAI 2015. `https://easychair.org/publications/paper/Q41W`.

[7] Connectors tend to be much simpler than general software and, therefore, are easier to synthesize.

2.2 DMM subclasses

A number of existing well-known formalisms can be considered as subclasses of DMMs.

In addition to conventional neural networks (which can be understood as consisting of single neurons, or as consisting of layers and modules [yielding compact architectures]), one can also name synthesis based on composition of unit generators (which is the standard approach in digital audio synthesis) and at least some forms of probabilistic programming.

Patterns of creating programs in those subclasses, and patterns of training/program synthesis in those subclasses can potentially be used in the context of general DMMs[8].

2.3 V-values as network matrices

We use 6-dimensional tensors as network matrices in our current implementation of DMMs based on V-values and variadic neurons. (We have also considered removing the activation function from that and making it a parameter of a neuron, with a possibility of using a linear combination of activation functions, which would lead to 4-dimensional tensors as network matrices.)

However, potentially we can use "mixed rank" tensors (general V-values) as network matrices (instead of "flat" tensors we are currently using for that).

We have defined a way to use a non-flat V-value as a multiplicative mask and as a replacement for a flat vector of coefficients in an operation of taking a linear combination of V-value subtrees[9].

This means that the subsequent `apply-matrix` function only needs to have a flat (fixed number of levels) structure of indices for matrix rows, but can use arbitrarily shaped V-values as matrix rows themselves in the current implementation (only the use patterns in the examples would need to change).

One might also want to allow non-flat structure on the "upper level", with matrix rows being leaves of a V-value. This does require explicitly allowing leaves of this kind (although a plain number can be subsumed as a V-value scalar). Informally, this requirement can be understood as the need to know where in the path to a leaf the row-related keys end and the column-related keys begin (currently we just rely on the convention that the first 3 keys in the path are row-related).

One would also need to add together the result of applying different matrix rows (e.g. one row can create a tree within a V-value, and another row can create its subtree, so the sum of these contributions would need to be taken). The present subsection is just a rough sketch, but can be made precise (we were sharing related informal notes within our group for a while now.)

[8] Methods based on oscillations and on spiking networks and spike synchronization also belong here.

[9] See functions `rec-map-mult-mask` and `rec-map-lin-comb`, lines 105-137 and 139-182 of `https://github.com/jsa-aerial/DMM/blob/master/src/dmm/core.clj`.

3 Diversity scenarios: populations and hybrids

We start with "editing running DMMs on the fly" series of experiments described in Section 1.1. There is no reason why the editor should be implemented in the same language or should run on the same computer as the DMM itself, hence we arrive at asynchronous exchange of V-values between different processes via one of several available mechanisms (e.g. websockets).

Populations. The next step is to consider a population of DMMs running at their own speed and capable of asynchronously exchanging V-values with each other, including "editing suggestions" (data, which the receiving network can transform into edits of its own network matrix). This would be particularly important in the next section covering learning methods (Section 4).

Hybrid populations. The next step to consider hybrid populations where conventional software coexist with DMMs (perhaps with different varieties of DMMs); all that is needed is to equip conventional software with the ability to emit and receive V-values from time to time. (If one needs to incorporate conventional software which performs work of finite duration in time [that is, transforming an input to an output and exiting], it might be convenient in this paradigm to wrap it into a layer which works indefinitely long and invokes from time to time the software which performs finite work.)

Discrete data. One way to incorporate discrete data in our framework is to include them into leaves as samples from signed probability distributions. This route is well explored in our publication. Another way is to consider formal finite linear combinations of discrete data in question[10]. One can either extend leaves to accommodate such linear combinations, or one can simply use map keys to represent the data of interest (and then the framework is unchanged)[11].

Multimedia streams. We need to make sure that we can exchange multimedia linear streams such as audio and animation. This places different requirements in terms of bandwidth and latency compared to asynchronous exchange of sparse streams of V-values. Initial experiments of injecting streams such as webcam feeds and loops of recorded videos were performed in recent months for pre-DMM[12] and Lightweight Pure DMM[13] architectures.

[10] The technical name for this construction is **vector space generated by a given set**. Sparsity conditions limiting the number of non-zero coefficients can be imposed on the level of implementation if needed.

[11] In general, the potential of using meaningful languages of the map keys is insufficiently explored by us. This potential is very interesting from many angles.

[12] `https://github.com/anhinga/fluid/tree/master/atparty-2018/surreal_webcam`

[13] `https://github.com/anhinga/fluid_drafts/tree/master/Lightweight_with_Movies`

4 Learning methods

Within hybrid populations, diverse learning methods can be present at the same time.

Lightweight Pure DMMs have regular structure and thus are well suited for traditional learning setups, with well-formalized differentiable objectives, batching, and GPU computations.

At the same time, DMMs based on V-values and variadic neurons which tend to have highly irregular dynamic structure are usually not well-covered by existing automated differentiation tools, and making them friendly for GPUs is an open problem at this time.

Therefore, **derivative-free methods** seem to be more attractive for DMMs based on V-values and variadic neurons at the moment, and *one hopes to gain learning speed in this context from the power of learning to learn methods,* rather than from the power of hardware, and also from the interactions with faster learners within the hybrid population.

Note that learning can be understood via replacing the vector of parameters $\mathbf{W}$ with $\mathbf{W}+\Delta\mathbf{W}$. Therefore, when we are talking about learning to learn, we should be talking of learning to produce $\Delta\mathbf{W}$. This calls for thinking in terms of formal differences of DMMs (neural networks, programs), $\mathbf{W}_2$-$\mathbf{W}_1$.

Populations of directions of change. In particular, we started to experiment with a simple version of population coordinate descent[14]. The population coordinate descent is the scheme of coordinate descent we proposed last year[15] where coordinates are sampled from an overdefined coordinate system and, generally speaking, the probability distribution over this set of coordinates can change in adaptive manner.

Populations of DMMs and other programs computing potentially useful directions of change. It is more natural to rate not the possible directions of change themselves, but the sources suggesting those directions, since the sources can be smart and take existing context into account.

Hence, the setup from Section 3 is natural: there is a population of sources coming up with individualized editing suggestions for other members of population. They are rated adaptively by recipients, depending on the suitability of their suggestions for a given recipient.

This is a promising line of reasoning, to be further developed.

Final remarks

Here are some additional remarks which might be useful during the next stage of DMM research and DMM-related software design and development.

[14] See `afterlife_balanced_coord_updates` in `https://github.com/anhinga/fluid/tree/master/Lightweight_Evolutionary`.

[15] `https://github.com/jsa-aerial/DMM/blob/master/design-notes/Early-2017/population-coordinate-descent.md`

Hierarchical DMMs. One motivation for possibly using general V-values as network matrices (Section 2.3) is that this enables grouping neurons into hierarchical structures without concatenating their names to fit the flat indexing system.

Deep copy of subgraphs We know matrix transformations for the style of creating complicated and possibly "pseudo-fractal" networks on the fly via deep copying of subgraphs[16]. Hierarchical DMMs should make this easier by allowing to define subgraphs via hierarchical groups of neurons.

Diversity of DMM viewing and editing interfaces. The considerations of Section 3 are calling for a system which is not monolithic, but consists of loosely connected parts, where it is easy to add new parts. Therefore we can have multiple interfaces for viewing and editing DMM structure on the fly, including interfaces based on graph visualization and visual editing.

[16]Section 4 of Michael Bukatin, Steve Matthews, Andrey Radul, **Programming Patterns in Dataflow Matrix Machines and Generalized Recurrent Neural Nets**, `https://arxiv.org/abs/1606.09470`