Higher-order neuromorphic computations with linear streams

Mishka (Michael Bukatin)

Dataflow Matrix Machines project

`https://anhinga.github.io`

CCC 2020: Continuity, Computability, Constructivity – From Logic to Algorithms

September 3, 2020

I overview work done in 2012-2020.

Most of this work is done in a series of research collaborations with Steve Matthews, Ralph Kopperman, Andrey Radul, Jon Anthony.

I am looking for collaborators.

e-mail: `bukatin@cs.brandeis.edu`

these slides are linked from
`https://www.cs.brandeis.edu/~bukatin/dmm_next.html`

1. How to make vector spaces from Scott domains
   - Add overdefined elements (to cancel partially defined elements)
   - Obtain bicontinuous domains with two Scott topologies
   - Obtain rich mathematical landscape
2. Computing with vector-like streams
   - Natural degree of generality for neural model of computation
   - Continuously deformable general-purpose programs
   - Neural machines which can fluently modify themselves
3. Interplay between 1 and 2, and open problems

1. How to make vector spaces from Scott domains
   - Add overdefined elements (to cancel partially defined elements)
   - Obtain bicontinuous domains with two Scott topologies
   - Obtain rich mathematical landscape

2. Computing with vector-like streams
   - Natural degree of generality for neural model of computation
   - Continuously deformable general-purpose programs
   - Neural machines which can fluently modify themselves

3. Interplay between 1 and 2, and open problems

1. How to make vector spaces from Scott domains
   - Add overdefined elements (to cancel partially defined elements)
   - Obtain bicontinuous domains with two Scott topologies
   - Obtain rich mathematical landscape

2. Computing with vector-like streams
   - Natural degree of generality for neural model of computation
   - Continuously deformable general-purpose programs
   - Neural machines which can fluently modify themselves

3. Interplay between 1 and 2, and open problems

The essence of problem, consider negation in interval numbers:

$$[5,7] + [-7,-5] = [-2,2] \sqsubset [0,0].$$

To fix this: allow **pseudosegments**,
where the endpoints are in the "wrong" order:

$[7,5], [-5,-7]$, etc.

Then we get **true negation** and an **abelian group**.

The essence of problem, consider negation in interval numbers:

$[5,7] + [-7,-5] = [-2,2] \sqsubset [0,0]$.

To fix this: allow **pseudosegments**,
where the endpoints are in the "wrong" order:

$[7,5], [-5,-7]$, etc.

Then we get **true negation** and an **abelian group**.

The essence of problem, consider negation in interval numbers:

$[5,7] + [-7,-5] = [-2,2] \sqsubset [0,0]$.

To fix this: allow **pseudosegments**,
where the endpoints are in the "wrong" order:

$[7,5], [-5,-7]$, etc.

Then we get **true negation** and an **abelian group**.

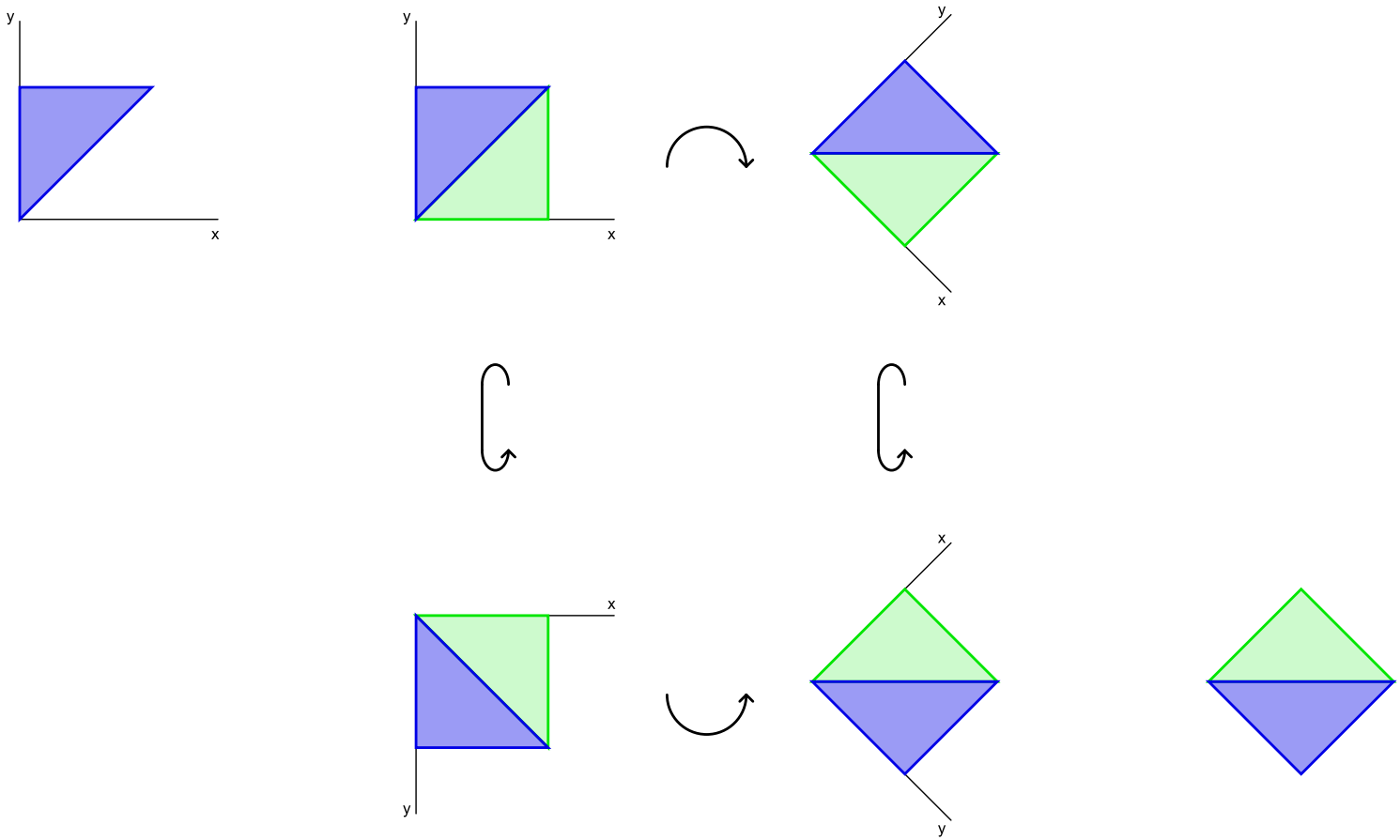
Multiple rediscoveries

Known under various names: Kaucher interval arithmetic, directed interval arithmetic, generalized interval arithmetic, modal interval arithmetic, interval algebraic extensions, etc.

First mention we know: M. Warmus, Calculus of Approximations. Bull. Acad. Pol. Sci., Cl. III, 4(5): 253-259, 1956, `http://www.cs.utep.edu/interval-comp/warmus.pdf`

A comprehensive repository of literature on the subject is maintained by Evgenija Popova: The Arithmetic on Proper & Improper Intervals (a Repository of Literature on Interval Algebraic Extensions), `http://www.math.bas.bg/~epopova/directed.html`

From Cartesian to Hasse representation



Bicontinuous domains

K. Keimel. Bicontinuous domains and some old problems in domain theory. *Electronic Notes in Theoretical Computer Science*, **257**:35-54, 2009

- Two Scott topologies, not one.
- Order-reversing involution.
- Computations can be done via either of these topologies.
- Use involution to switch between them?
- Classical logic (classical treatment of negation)?

Partial inconsistency landscape

- Negative distance/probability/degree of set membership
- Bilattices
- Partial inconsistency
- Non-monotonic inference
- Bitopology
- $x = (x \wedge 0) + (x \vee 0)$ or $x = (x \wedge \bot) \sqcup (x \vee \bot)$
- Scott domains tend to become embedded into vector spaces
- Modal and paraconsistent logic and possible world models
- Bicontinuous domains
- The domain of arrows, $D^{Op} \times D$ or $C^{Op} \times D$

M. Bukatin, S. Matthews. Linear Models of Computation and Program Learning. In G. Gottlob, G. Sutcliffe, A. Voronkov, editors, GCAI 2015, *EasyChair Proceedings in Computing*, **36**, 66-78. `https://easychair.org/publications/paper/Q4lW`

M. Bukatin. *Using streams of probabilistic samples in neural machines.* January 2020.
`https://github.com/anhinga/2020-notes/tree/master/research-notes`

We have rich mathematics of bicontinuous domains, which seems likely to be suitable to define denotational **vector semantics** of programming languages.

The natural question then arises: can one program in terms of vector-like elements?

That is, can one define **operational semantics** (implementation) in terms of vector-like elements?

More specifically, is programming with **linear streams** (streams which can be combined with numerical coefficients) sufficient for general-purpose programming?

We'll be talking about programming in **dataflow style** (or, in more modern terms, in **functional reactive style**).

We know that short **probabilistic programs** are very expressive.

So are short **functional reactive animations programs** generalizing digital audio synthesis by composition of unit generators, see e.g. `https://youtu.be/fEWcg_A5UZc`

Our present collection of programming techniques and examples is diverse enough to support the claim that this formalism is sufficiently expressive for general purpose programming:

M. Bukatin. *Overview of programming techniques and examples in DMM literature.* January 2020.

`https://github.com/anhinga/2020-notes/tree/master/programming-overview`

But can one do higher-order programming in this style? And can one **deform the resulting programs in a continuous fashion**?

It turns out that the answer is yes, if one agrees to **interleave linear and non-linear transformations of linear streams**.

Here we came from the viewpoint of stream-oriented programming.

One can also arrive at the same result from the viewpoint of **recurrent neural networks**.

The essence of neural model of computations is that linear and non-linear computations are interleaved. So, the natural degree of generality for neuromorphic computations is to work not with streams of numbers, but with arbitrary **linear streams**.

Dataflow matrix machines (DMMs)

DMMs: a novel class of **neural machines**.

They use arbitrary **linear streams** instead of streams of numbers.

- **Self-modification:** can fluently modify their own weights, connectivity patterns, and size;
- Highly expressive linear streams of V-values (vector-like values based on nested dictionaries);
- Expressive enough to serve as a programming platform: functional reactive programming with **continuously deformable programs**.

Linear streams

The key feature of **DMMs** compared to **RNNs**: they use **linear streams** instead of streams of numbers.

The following streams all support the pattern of alternating linear and non-linear computations:

- Streams of numbers
- Streams of vectors from fixed vector space V
- Linear streams: such streams that the notion of **linear combination of several streams** is defined.

Kinds of linear streams

Some examples:

- Every vector space V gives rise to the corresponding kind of linear streams (streams of vectors from that space)
- Every measurable space X gives rise to the space of **streams of probabilistic samples** drawn from X and decorated with $+/-$ signs (linear combination is defined by a stochastic procedure)
- Streams of images of a particular size (that is, animations)
- Streams of matrices; streams of multidimensional arrays
- Streams of V-values based on nested maps (instead of S-expressions)

V-values: vector space based on nested dictionaries

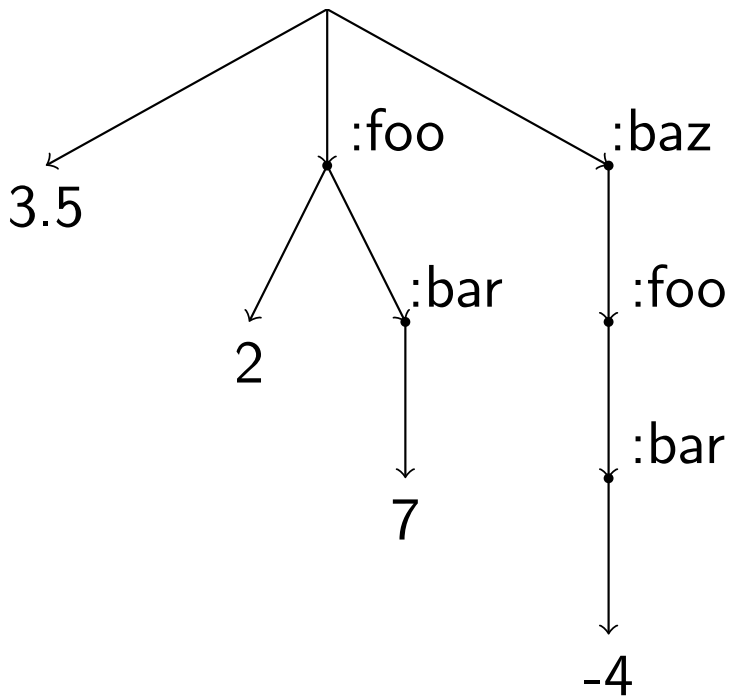
V-values play the role of Lisp S-expressions in this formalism.

We want a vector space.

Take prefix trees with numerical leaves
implemented as nested dictionaries.

We call them **V-values** ("vector-like values").

Example of a V-value: different ways to view it

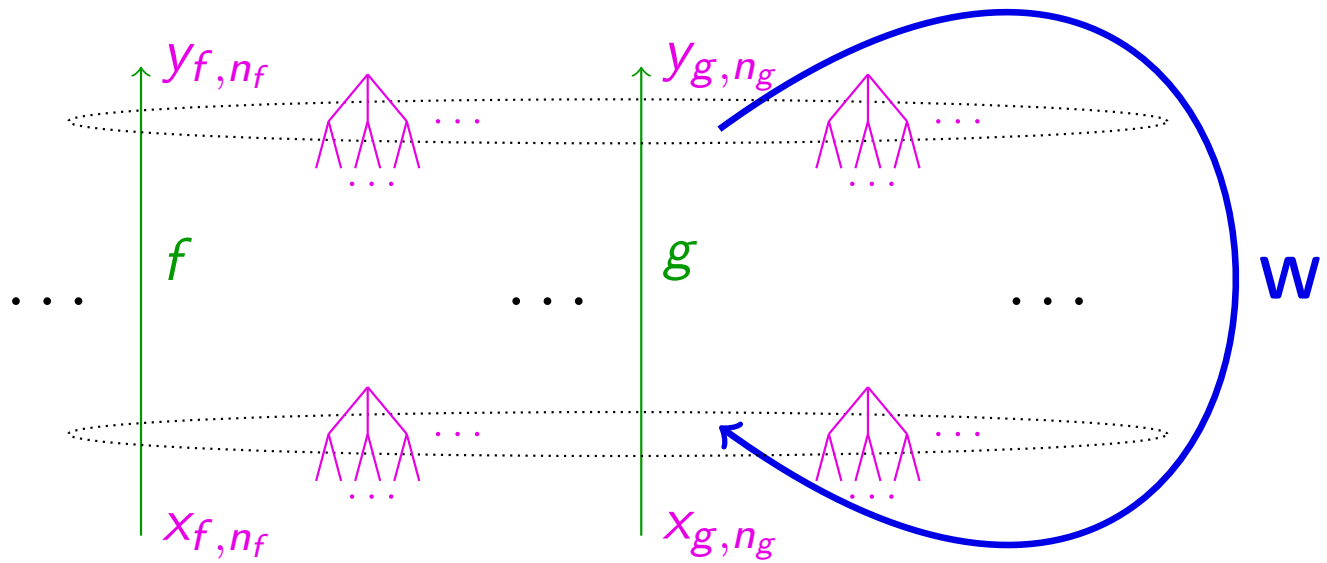


- $3.5 \cdot (\epsilon) + 2 \cdot (\text{:foo}) +$
  $7 \cdot (\text{:foo :bar}) - 4 \cdot (\text{:baz :foo :bar})$
- $(\rightsquigarrow 3.5) + (\text{:foo} \rightsquigarrow 2) +$
  $(\text{:foo} \rightsquigarrow \text{:bar} \rightsquigarrow 7) +$
  $(\text{:baz} \rightsquigarrow \text{:foo} \rightsquigarrow \text{:bar} \rightsquigarrow -4)$
- scalar 3.5 + sparse 1D array `{d1[:foo]= 2}` +
  sparse 2D matrix `{d2[:foo, :bar]= 7}` +
  sparse 3D array `{d3[:baz, :foo, :bar]= -4}`
- `{:number 3.5,`
  `:foo {:number 2, :bar 7},`
  `:baz {:foo {:bar -4}}}`
  $(\texttt{:number} \notin L)$

Dataflow matrix machines (our current implementation) based on streams of V-values and variadic neurons

$$x^{t+1}_{f,n_f,i} = \sum_{g \in F} \sum_{n_g \in L} \sum_{o \in L} w^t_{f,n_f,i;\,g,n_g,o} * y^t_{g,n_g,o} \text{ (down movement)}$$

$$y^{t+1}_{f,n_f} = f(x^{t+1}_{f,n_f}) \text{ (up movement)}$$



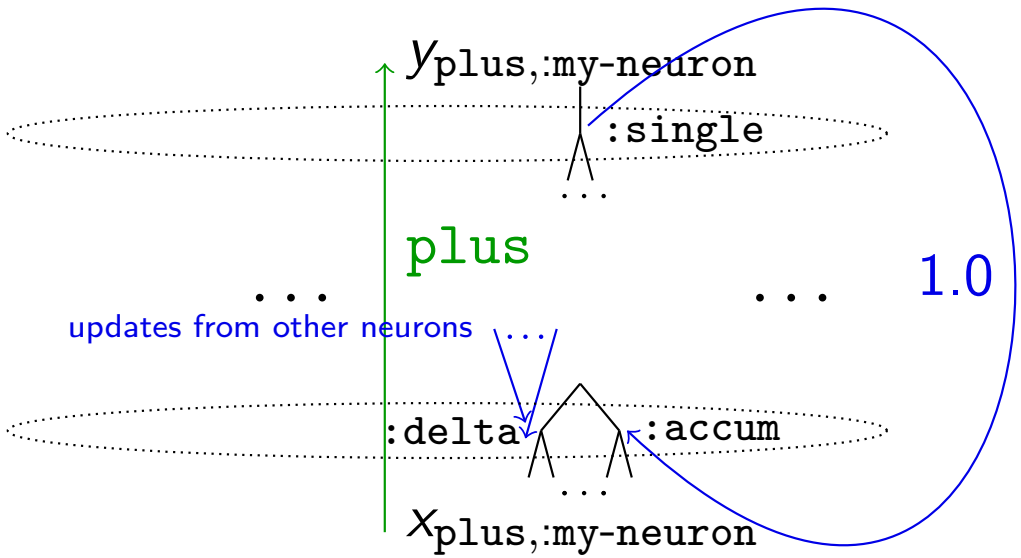
DMMs: programming with powerful neurons

Powerful variadic neurons and streams of V-values $\Rightarrow$ a much more expressive formalism than networks based on streams of numbers.

Many tasks can be accomplished by compact DMM networks, where **single neurons function as layers or modules**.

- Accumulators and memory
- Multiplicative constructions and "fuzzy if"
- Sparse vectors
- Data structures
- Self-referential facilities

Accumulators and memory



In this implementation, activation function `plus` adds V-values from `:accum` and `:delta` together and places the result into `:single`.

Sparse vectors of high or infinite dimension

Example: a neuron accumulating count of words in a given text.

The dictionary mapping words to their respective counts is an infinite-dimensional vector with a finite number of non-zero elements.

- Don't need a neuron for each coordinate of our vector space.
- Don't have an obligation to reduce dimension by embedding.

Streams of immutable data structures

One can represent **lists, matrices, graphs,** and so on via nested dictionaries.

It is natural to use streams of immutable V-values in the implementations of DMMs.

The DMM architecture is friendly towards algorithms working with immutable data structures in the spirit of functional programming.

But more imperative styles can be accommodated as well.

Self-referential facilities (easy in DMMs)

It is easy to represent the network matrix **W** as a V-value.

Emit the stream of network matrices from neuron `Self`.

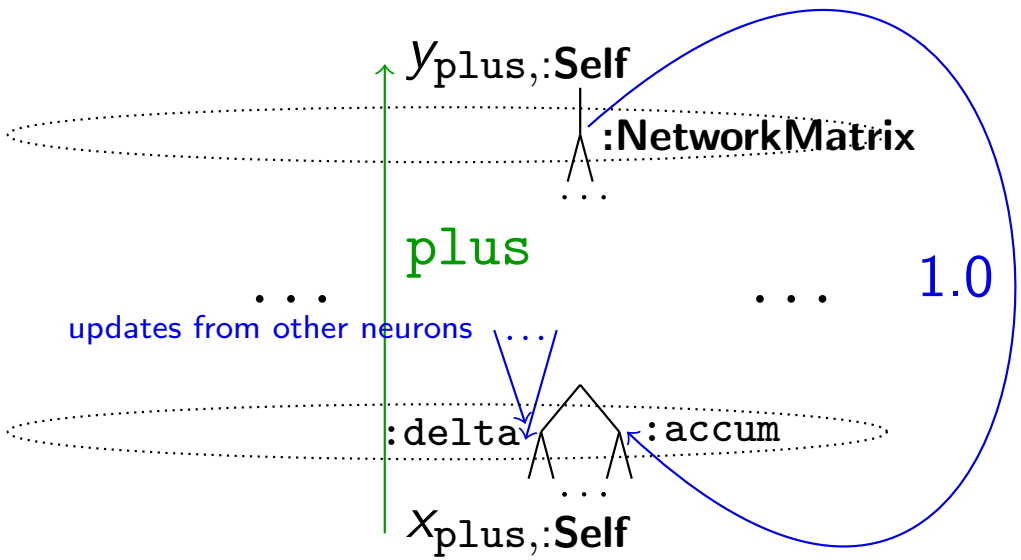
Use the most recent V-value from that stream as the network matrix **W** during the next "down movement".

This mechanism allows a DMM network to **modify its own weights, topology, and size** while it is running.

Self-referential facilities (easy in DMMs)

Our current implementation: `Self` connected as an accumulator.

It accumulates the value of the network matrix and accepts additive updates from other neurons in the network.



The most recent V-value at the `:NetworkMatrix` output of $y_{\texttt{plus},:\textbf{Self}}$ neuron is used as **W**.

Monotonic evolution of Warmus numbers by additions

Consider $x \sqsubseteq (x + x_1) \sqsubseteq (x + x_1 + x_2) \sqsubseteq \ldots$

Then every $x_i = [a_i, b_i]$ must be a pseudo-segment anti-approximating zero:

$[0, 0] \sqsubseteq [a_i, b_i]$, that is $b_i \leq 0 \leq a_i$.

(`https://arxiv.org/abs/1610.00831`, Appendix A)

Rectifiers and quasi-metrics

It was typical to use sigmoid non-linearities as activation functions in neural nets, but a few years ago people discovered that **ReLU** (rectified linear units) often work much better:
$\mathbf{ReLU}(x) = max(0, x)$.

This is an integral of the Heaviside step function. Lack of smoothness at 0 does not seem to interfere with gradient methods, and otherwise it's nice when the derivatives are so simple.

Our standard quasi-metrics on reals are closely related to ReLU:

$q_1(x, y) = \mathbf{ReLU}(x - y) = q_2(y, x)$.

(`https://arxiv.org/abs/1610.00831`, Appendix B)

Open Problems

There are a bit more known connections between bicontinuous domains and DMM practice, but there should be much more.

Usually, we forget that programming with linear streams is related to bicontinuous domains, and we just use vector spaces, and we lose the potential hidden in the math of partial inconsistency.

We do have some hints. E.g. training of **generative adversarial networks** does resemble computations with two Scott topologies and involution in spirit. One of the open problems is to investigate whether this connection can be formalized.

Open Problems

Both Klaus Keimel and Dexter Kozen have beginnings of higher-order bicontinuous domain theory in their respective papers, but bicontinuous domain equations are not done.

Now we finally have a self-referential formalism on the level of operational semantics, so this might provide guidance towards figuring out the adequate theory of domain equations in this case.

A lot of open problems, theoretical and applied, are collected here:

M. Bukatin. *Dataflow matrix machines: a collaborative research agenda.* August 2020.

`www.cs.brandeis.edu/~bukatin/dmm-collaborative-research-agenda.pdf`

Open Problems

Bringing together DMMs and Probabilistic Programming:

- Using negative probabilities and streams of signed samples in probabilistic programming;
- Using DMMs as probabilistic programs;
- Using DMMs as samplers.

M. Bukatin. *Using streams of probabilistic samples in neural machines.* January 2020.

`https://github.com/anhinga/2020-notes/tree/master/research-notes`

Open Problems

We have recently started to investigate connections between DMMs and Transformers.

- Could what we know about DMMs shed some light on the remarkable properties of Transformers?
- What are the ways to incorporate key elements from Transformer architecture into a more flexible DMM setup, and, in particular, could we obtain interesting compact and low training cost models by incorporating attention-inspired and Transformer-inspired motives into DMMs?

See here for more details:

M. Bukatin. *Dataflow matrix machines: a collaborative research agenda.* August 2020.

References

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

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

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

Mishka on GitHub: `https://github.com/anhinga`

e-mail: `bukatin@cs.brandeis.edu`

these slides are linked from
`https://www.cs.brandeis.edu/~bukatin/dmm_next.html`.