Dataflow Matrix Machines and V-values: a Bridge between Programs and Neural Nets

Michael Bukatin

HERE Technologies

Joint work with Jon Anthony (Boston College)

- - -

IBM AI Systems Day
Cambridge, MA, October 3, 2018

Dataflow matrix machines (DMMs)

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

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

Use of linear streams by single neurons is the main source of their power compared to RNNs. The extra power also comes from:

- Arbitrary fixed or variable arity of neurons;
- Highly expressive linear streams of V-values (vector-like values based on nested dictionaries);
- Unbounded network size ($\Rightarrow$ unbounded memory);
- Self-referential facilities: ability to change weights, topology, and the size of the active part dynamically, on the fly.

DMMs vs RNNs as a programming formalism

One of the key motivations for this work was to create a "neuromorphic programming formalism".

- Neural nets: gap between theory and practice
  - Theory says we should be able to use RNNs as a programming formalism, because they are Turing-complete.
  - In practice, RNNs are not expressive enough for that.
- DMMs are powerful enough to write programs. Therefore, we obtain a programming framework where
  - One can **deform programs in continuous manner**.
  - A program is determined by a matrix of numbers. Synthesize a matrix of numbers to synthesize a program.

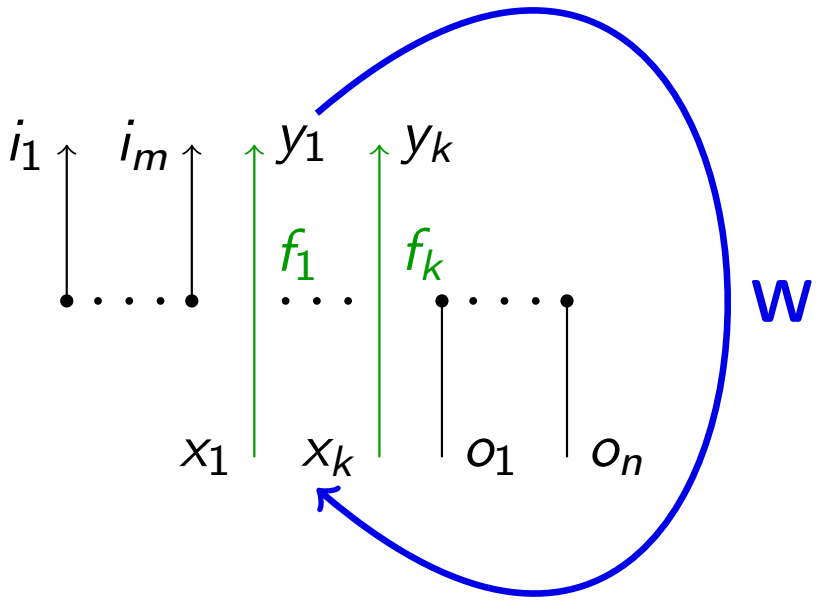
Neural nets: gap between theory and practice

Andrej Karpathy: "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."

`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Edward Grefenstette, **Limitations of RNNs: a computational perspective**, NAMPI 2016 workshop at NIPS.

RNNs: linear and non-linear computations are interleaved



"Two-stroke engine" for an RNN:

"Down movement" (linear[1]):

$$(x_1^{t+1}, \ldots, x_k^{t+1}, o_1^{t+1}, \ldots, o_n^{t+1})^\top = \mathbf{W} \cdot (y_1^t, \ldots, y_k^t, i_1^t, \ldots, i_m^t)^\top.$$

"Up movement" (non-linear[2]):

$$y_1^{t+1} = f_1(x_1^{t+1}), \ldots,$$
$$y_k^{t+1} = f_k(x_k^{t+1}).$$

[1] linear and global in terms of connectivity

[2] usually non-linear, local

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

The notion of **linear streams** is more general than the notion of **streams of vectors**. 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 (will revisit)

Dataflow matrix machines (continued)

Countable network with finite active part at any moment of time.

Countable matrix **W** with finite number of non-zero elements at any moment of time. Unbounded memory for Turing completeness.

"Down movement":
For all inputs x_{i,C_k} where there is a non-zero weight $w^t_{(i,C_k),(j,C_m)}$:

$$x^{t+1}_{i,C_k} = \sum_{\{(j,C_m)|w^t_{(i,C_k),(j,C_m)}\neq 0\}} w^t_{(i,C_k),(j,C_m)} * y^t_{j,C_m}.$$

"Up movement":
For all active neurons C:

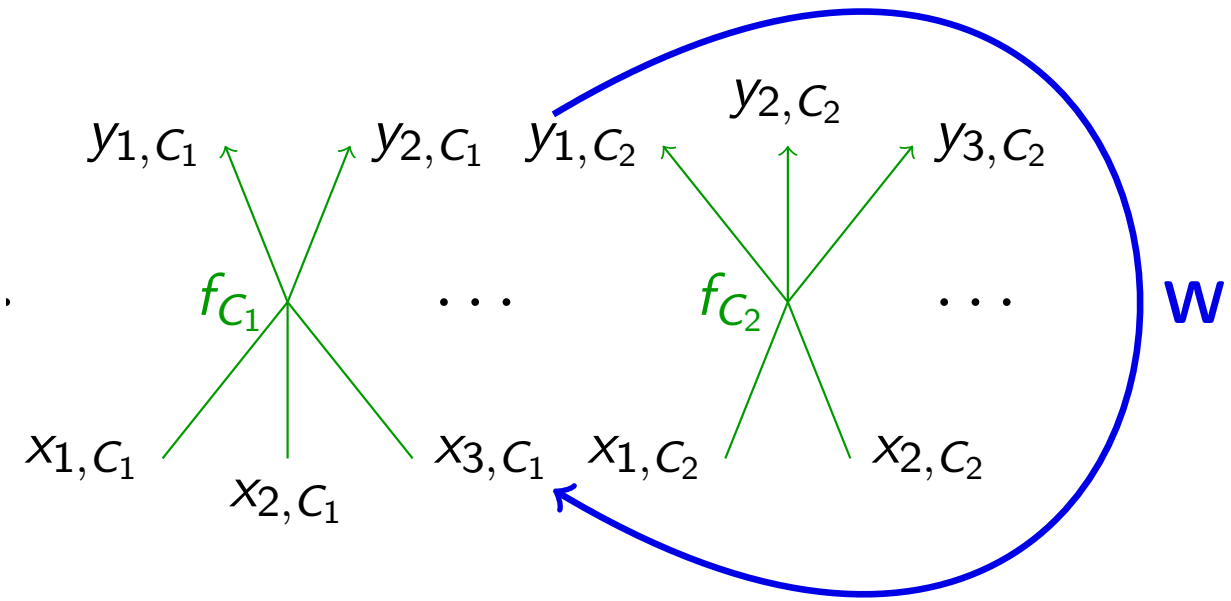
$$y^{t+1}_{1,C}, ..., y^{t+1}_{p,C} = f_C(x^{t+1}_{1,C}, ..., x^{t+1}_{n,C}).$$

Here x_{i,C_k} and y_{j,C_m} are linear streams.
Neurons in DMMs have arbitrary arity!

Dataflow matrix machines (continued)

Countable network with finite active part at any moment of time.

Countable matrix **W** with finite number of non-zero elements at any moment of time.



"Down movement":
For all inputs x_{i,C_k} where there is a non-zero weight $w^t_{(i,C_k),(j,C_m)}$:

$$x^{t+1}_{i,C_k} = \sum_{\{(j,C_m)|w^t_{(i,C_k),(j,C_m)}\neq 0\}} w^t_{(i,C_k),(j,C_m)} * y^t_{j,C_m}.$$

"Up movement":
For all active neurons C:

$$y^{t+1}_{1,C}, ..., y^{t+1}_{p,C} = f_C(x^{t+1}_{1,C}, ..., x^{t+1}_{n,C}).$$

Here x_{i,C_k} and y_{j,C_m} are linear streams.
Neurons in DMMs have arbitrary arity!

Type correctness condition for mixing different kinds of linear streams in one DMM

Type correctness condition: $w^t_{(i,C_k),(j,C_m)}$ is allowed to be non-zero only if x_{i,C_k} and y_{j,C_m} belong to the same **kind** of linear streams.

Next: linear streams of **V-values** based on **nested dictionaries**:

- sufficiently universal and expressive to save us from the need to impose type correctness conditions.
- allow us to define **variadic neurons**, so that we don't need to keep track of input and output arity either.

V-values: vector space based on nested dictionaries

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

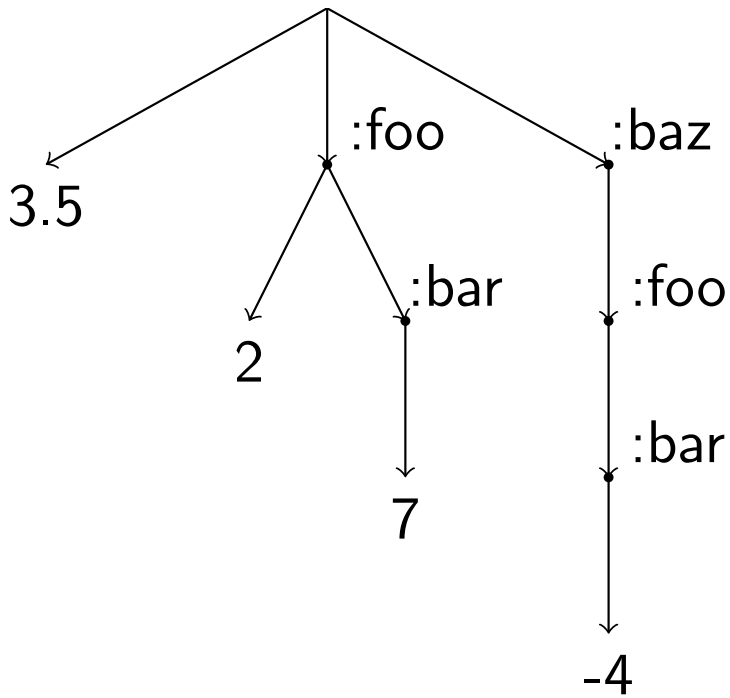
We want a vector space.

Take prefix trees with numerical leaves
implemented as nested dictionaries.

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

(A more general construction of V-values with "linear stream" leaves: Section 5.3 of `https://arxiv.org/abs/1712.07447`)

Example of a V-value



- $3.5 \cdot (\epsilon) + 2 \cdot (\text{:foo}) +$
  $7 \cdot (\text{:foo :bar}) - 4 \cdot (\text{:baz :foo :bar})$
- $(\rightsquigarrow 3.5) + (\text{:foo} \rightsquigarrow 2) +$
  $(\text{:foo} \rightsquigarrow \text{:bar} \rightsquigarrow 7) +$
  $(\text{:baz} \rightsquigarrow \text{:foo} \rightsquigarrow \text{:bar} \rightsquigarrow -4)$
- 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}}}`
  $(\texttt{:number} \notin L)$

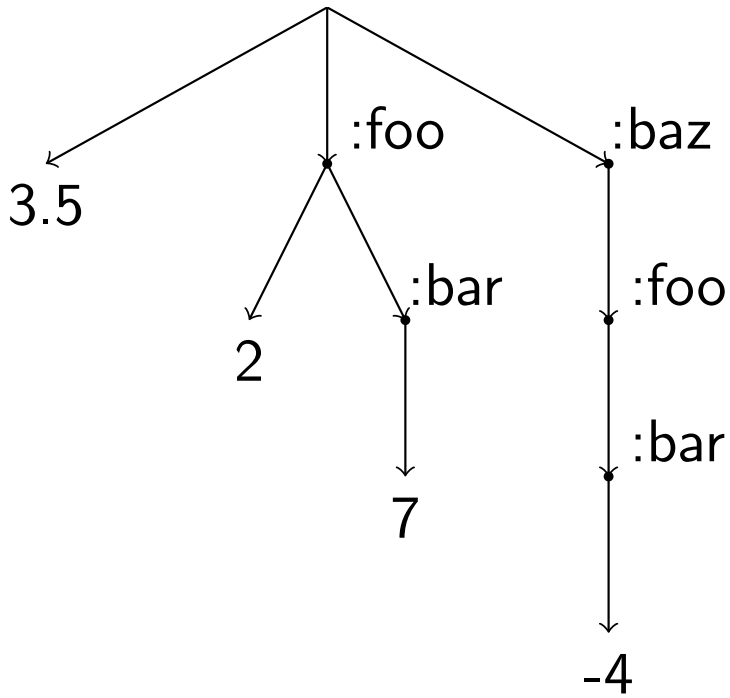
Ways to understand V-values

Consider ordinary words ("labels") to be letters of a countable "super-alphabet" L.

V-values can be understood as

- Finite linear combinations of finite strings of letters from L;
- Finite prefix trees with numerical leaves;
- Sparse "tensors of mixed rank" with finite number of non-zero elements;
- Recurrent maps (that is, nested dictionaries) from $V \cong \mathbb{R} \oplus (L \to V)$ admitting finite descriptions.

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

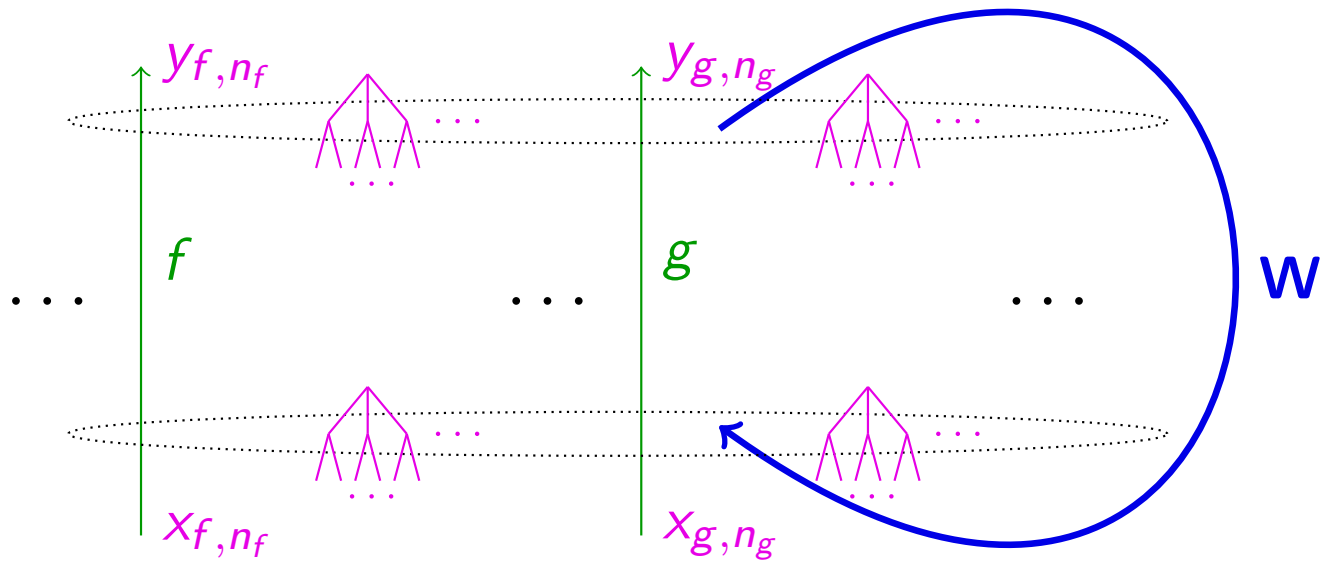


- $3.5 \cdot (\epsilon) + 2 \cdot (:\text{foo}) +$
  $7 \cdot (:\text{foo}\ :\text{bar})$ - $4 \cdot (:\text{baz}\ :\text{foo}\ :\text{bar})$
- $(\rightsquigarrow 3.5) + (:\text{foo} \rightsquigarrow 2) +$
  $(:\text{foo} \rightsquigarrow :\text{bar} \rightsquigarrow 7) +$
  $(:\text{baz} \rightsquigarrow :\text{foo} \rightsquigarrow :\text{bar} \rightsquigarrow \text{-}4)$
- 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

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

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



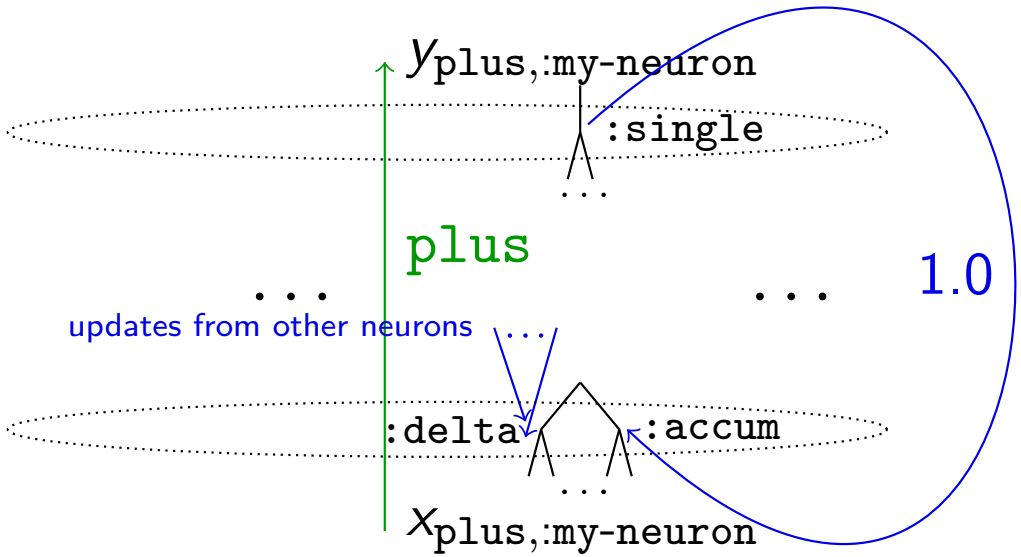
DMMs: programming with powerful neurons

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`.

Multiplicative masks and fuzzy conditionals (gating)

Many multiplicative constructions enabled by **input arity** > 1.

The most notable is multiplication of an otherwise computed neuron output by the value of one of its scalar inputs.

This is essentially a **fuzzy conditional**, which can

- selectively turn parts of the network on and off in real time via multiplication by zero
- attenuate or amplify the signal
- reverse the signal via multiplication by -1
- redirect flow of signals in the network
- ...

- (Used implicitly in LSTM and Gated Recurrent Unit networks.)

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

Neural networks: again a gap between theory and practice.

Theory:
Jurgen Schmidhuber, **A 'self-referential' weight matrix**, 1993.

Practice: ???

Difficult to do well with streams of scalars because of dimension mismatch: typically one has $\sim N$ neurons and $\sim N^2$ weights.

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`.

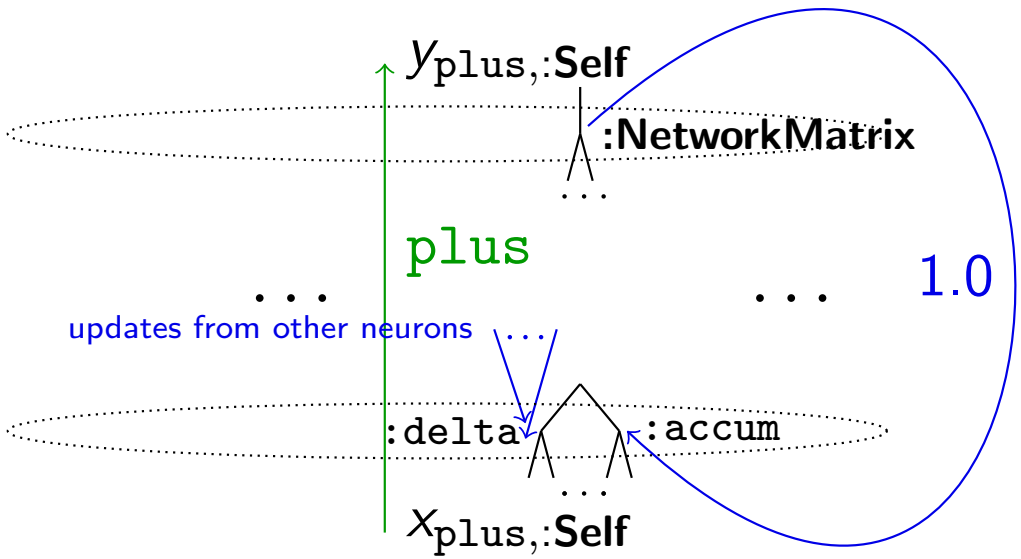
Use the most recent V-value from that stream as the network matrix **W** during the next "down movement".

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**.

Self-referential facilities (easy in DMMs)

Other neurons can use `Self` outputs to take into account the structure and weights of the current network **(reflection)**.

We have used self-referential mechanism to obtain waves of connectivity patterns propagating within the network matrix.

We have observed interesting self-organizing patterns in self-referential networks.

We also use this mechanism for "pedestrian purposes": to allow a user to edit a running network on the fly.

Future applications

- **Learning to learn**

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

- **Program synthesis**

  DMMs combine

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

- and more...

To recap:

Dataflow matrix machines use **linear streams** and not just streams of numbers.

This is the main source of their power. The extra power also comes from:

- Arbitrary fixed or variable arity of neurons;
- Highly expressive streams of V-values;
- Unbounded network size;
- Self-referential facilities (ability to change weights, topology, and the size of the active part dynamically).

References

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

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