Let's think about embedding neural machines into modulating fields

Mishka (Michael Bukatin)

`github:anhinga`

Dataflow Matrix Machines project

Longevity, AI, and Cognitive Research Hackathon
Cambridge, MA, October 27, 2024

Case for the role of electromagnetic fields in neuro

- **Michael Levin:**
  - **Strong case: electromagnetic fields are crucial in biological computations**
- Andrés Gómez-Emilsson and others:
  - Conjecture: qualia fields are carried by electromagnetic fields
- Joscha Bach and others:
  - Conjecture: bodies are antennas (receive and transmit)
- This all fits together, would explain a lot

Two routes

- Biorealistic
  - Realistic neural nets and electromagnetic fields
  - Even understanding how *C elegans* works would be nice
  - Very important, quite difficult
- Artificial architectures and “artificial fields”
  - Inspired by how we do “artificial attention”
  - Small-scale efforts have good chances for progress
  - **Needs a guiding light (what are we looking for?)**

Biological attention vs "artificial attention"

- Biological attention
    - Synchronized oscillations (*some* of the gamma waves)
    - Crick-Koch "40hz conjecture" (30hz-70hz)
        - "Towards a neurobiological theory of consciousness" (1990)
- "Artificial attention"
    - Linear combinations of "feature vectors"
    - Forget about the biological mechanism; just model the effects
        - Some features are emphasized, others are suppressed
    - Surprisingly effective, especially in hierarchies (Transformers)

“Artificial modulation”: what should be the guiding light?

What should be the guiding light for modulation of artificial neural machines by artificial fields?

Effectiveness in various tasks
(better training, better fine-tuning, better inference).

The artificial fields can be

- induced by neural machines themselves
- external
- a mix of both

One can modulate network inputs, network outputs, network connectivity weights, and other parameters (e.g. parameters of “activation functions”)

This has been the new part for this hackaton

Now I am going to tell you something which has been known before which is likely to be helpful if one wants to attack this problem.

What's the full generality here?

The essence of neural model of computations is that
linear and non-linear computations are interleaved.

What is the most general (and most flexible) way
to make this linear/non-linear alternating pattern possible?

. . . .

The natural degree of generality for neural-like computations is:

~~use only streams of numbers~~
use any streams supporting the notion of
linear combination of several streams ("linear streams")

Dataflow matrix machines (DMMs)

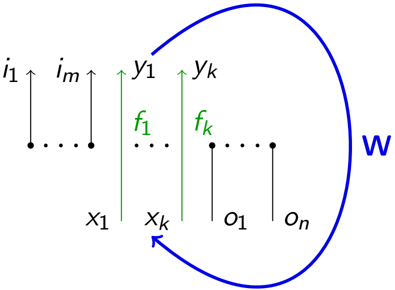
DMMs: a very natural class of **neural machines**.

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 (tree-shaped flexible tensors);
- 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.

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.

"Artificial attention":
taking linear combinations of high-dimensional objects.

Kinds of linear streams

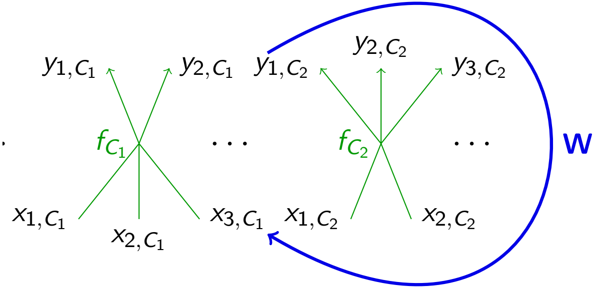
The notion of **linear streams** is more general than the notion of **streams of vectors**. Some examples:

- 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 (trees)

Dataflow matrix machines (DMMs, 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:

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

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

Skip to slide 27

The slides 14-26 are here just for convenience
if one wants to ponder this further later.

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`)

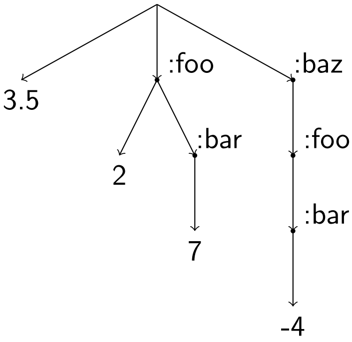
Ways to understand V-values

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

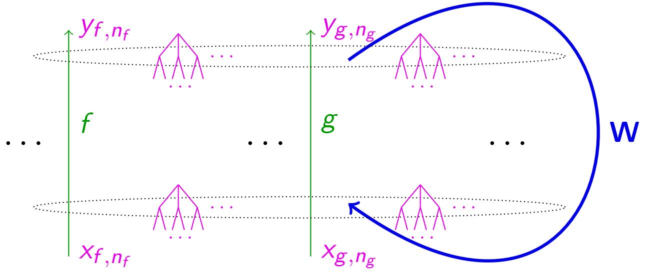


- $3.5 \cdot (\epsilon) + 2 \cdot$ (:foo) $+$
  $7 \cdot$ (:foo :bar) - $4 \cdot$ (:baz :foo :bar)
- ($\rightsquigarrow$ 3.5) $+$ (:foo $\rightsquigarrow$ 2) $+$
  (:foo $\rightsquigarrow$:bar $\rightsquigarrow$ 7) $+$
  (:baz $\rightsquigarrow$:foo $\rightsquigarrow$:bar $\rightsquigarrow$ -4)
- scalar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

Dataflow matrix machines (DMMs) based on streams of V-values and variadic neurons

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

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



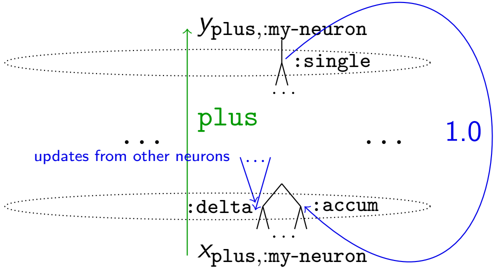
DMMs: programming with powerful neurons

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

$\Rightarrow$ 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
- ...

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

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

Easy in DMMs:

It is easy to represent the network matrix $\mathbf{W}$ as a V-value.

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

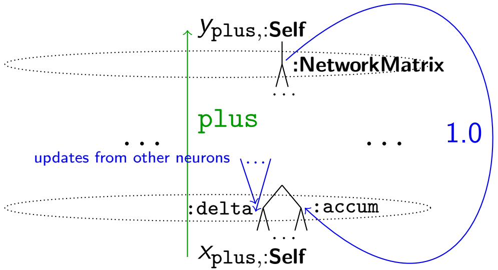
Use the most recent V-value from that stream as the network matrix $\mathbf{W}$ during the next "down movement".

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

Self-referential facilities

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 $\mathbf{W}$.

Self-referential facilities

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.

AI safety issues are quite real here

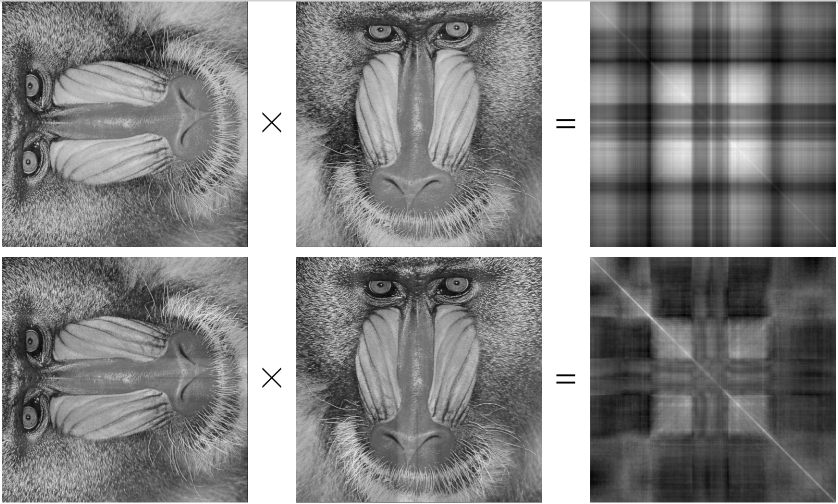
What should one use initially?

A GPU-friendly version with streams of matrices or other tensors.

And one should explore visually.

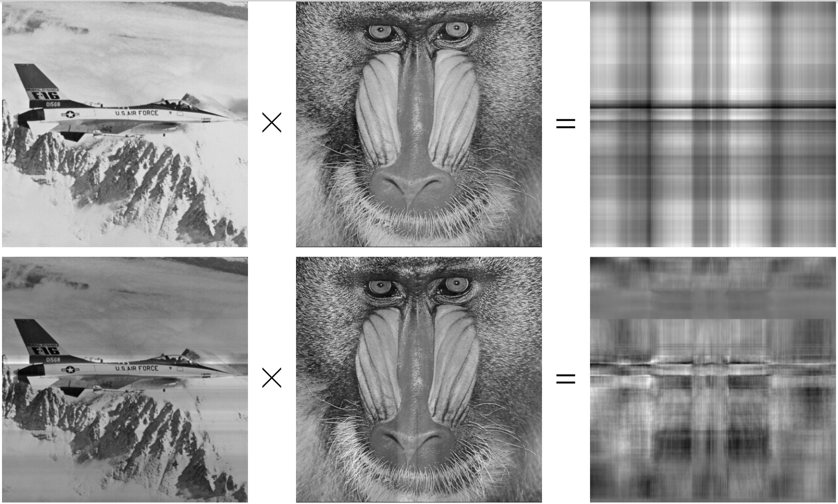
A few final remarks to help one explore visually.

For example, one can interpret monochrome images as matrices and multiply them as matrices.



In Transformers people sometimes **softmax** rows of the left matrix:
Attention$(Q, K, V) = \text{softmax}(cKQ^T)V$ from "Attention Is All You Need" (2017).

In the second example we **softmax** rows of the left matrix **and** columns of the right matrix resulting in products with richer, more fine-grained structure.



In Transformers people sometimes **softmax** rows of the left matrix:
Attention$(Q, K, V) = \text{softmax}(cKQ^T)V$ from "Attention Is All You Need" (2017).

In the second example we **softmax** rows of the left matrix **and** columns of the right matrix resulting in products with richer, more fine-grained structure.

Visual synthesis in the style of digital audio synthesis

Digital audio synthesis has been almost always done via **composition of unit generators**.

This style was invented by Max Mathews at Bell Labs in 1957.

`https://en.wikipedia.org/wiki/Max_Mathews`

One should think about those compositions of unit generators as handcrafted, hand-tuned custom neural machines.

E.g. a typical neuron might have this activation function: $f(a, b, x) = \sin(a * x + b)$.

Here we are talking about synthesis of images and animations in the same style, but with **high-dimensional flows** instead of flows of numbers.

Contact info

Contact: `github:anhinga` (e.g. open an issue)

bukatin@cs.brandeis.edu (or michael.bukatin at gmail)

I am looking for collaborations