# Dataflow matrix machines:
# - duality between **W** and its input;
# - shader-style non-linear component

Michael Bukatin

June 21, 2019

**Abstract**

We look at the intriguing natural duality between **W** and **Y**. This might eventually result in novel self-referential mechanisms in DMMs.

We also note that thinking about the non-linear part of the network as a shader might be quite fruitful.

It was observed before that compositional pattern-producing networks have shader architecture.

## Introduction

This research note emerged as a side effect of our design efforts and preliminary code explorations for the next generation of DMMs[1]. One particular focus of this new effort is that one would like to retain the full flexibility of DMMs, but at the same time to be able to efficiently train and run them on GPUs.

## 1 Duality in W·Y

We consider a standard two-stroke cycle neural machine with vector flows. The two-stroke cycle has a linear part controlled by matrix **W** and a non-linear part (traditionally understood as "activation functions of neurons do the work").

We denote the non-linear part as **F**, so the overall cycle of the machine is **F**∘**W** (or one might prefer **W**∘**F**, but we have found it more convenient to apply **W** first, so our standard view is **F**∘**W**; both views are quite useful).

Consider the flavor where the vectors in question have a fixed finite dimension and are shaped to form one-dimensional arrays.

The linear part applies **W** to the vector of vectors (data produced by the action of the non-linear part). This vector of vectors forms matrix **Y**. The application of **W** to this vector of vectors is simply matrix multiplication, **W**·**Y**.

The obvious duality here is that there is no particular way to distinguish between $\mathbf{W}$ and $\mathbf{Y}$, especially if both are generated by the action of the non-linear part, as typical for DMMs. If one wants to preserve the convention that the network matrix is on the left, all one needs to do for the duality is to transpose the whole thing: $\mathbf{X}^\top = \mathbf{Y}^\top \cdot \mathbf{W}^\top$.

## 2 Rectangular W vs. square W

There is tension in the field between the desire to view $\mathbf{W}$ as a square matrix and the desire to view it as a rectangular matrix.

$\mathbf{W}$ as a square matrix brings with itself all wonderfully convenient things, like eigenvalues, and being able to study the resulting dynamic systems by simply expecting $\mathbf{F}$ to have properties good enough to not interfere much compared to the case $\mathbf{F} = \mathbf{Id}$ (the case where $\mathbf{F}$ can be completely ignored).

E.g. a nice recent study introducing *recurrent identity networks*[2] represents $\mathbf{W}$ as $\mathbf{U}+\mathbf{Id}$, and then when one applies traditional regularization methods to elements of $\mathbf{U}$ instead of elements of $\mathbf{W}$, the problem of vanishing gradients mostly disappears. (If one would like to progress from empirical observations to mathematical theorems, one would still need to consider the $\mathbf{F}\circ\mathbf{W}$ or $\mathbf{W}\circ\mathbf{F}$ and to explicitly formulate some constraints on the properties of $\mathbf{F}$ needed for this consideration to hold.)

In this study we think about $\mathbf{W}$ as a rectangular matrix (even in those cases when $M$ and $N$ might accidentally coincide).

This will make it easier to emphasize asymmetries between inputs and outputs (both of $\mathbf{W}$ and of $\mathbf{F}$).

## 3 Redefining F and the two-stroke cycle

Let's say that $\mathbf{W}$ is $M\times N$ matrix, and $\mathbf{Y}$ is $N\times D$ matrix.

Here $D$ is the dimensionality of the vectors in our flows, $N$ is the number of vectors produced by the non-linear part, and $M$ is the number of vectors used as the input to the non-linear part. That is, the input to the non-linear part $\mathbf{F}$ is the $M\times D$-dimensional matrix $\mathbf{X} = \mathbf{W}\cdot\mathbf{Y}$.

It is time to stop thinking about the two-stroke cycle as having a linear part $\mathbf{W}$: $\mathbf{Y} \mapsto \mathbf{X} = \mathbf{W} \cdot \mathbf{Y}$, and to replace it with a bilinear part $\mathbf{B}$: $\langle \mathbf{W}, \mathbf{Y} \rangle \mapsto \mathbf{X}$.

Then, instead of $\mathbf{F}$: $\mathbf{X} \mapsto \mathbf{Y}$, we are going to have $\mathbf{F}$: $\mathbf{X} \mapsto \langle \mathbf{W}, \mathbf{Y} \rangle$.

Under this new way of thinking, the canonical view of the two-stroke cycle, $\mathbf{F}\circ\mathbf{B}$, maps $\mathbb{R}^{MN+ND}$ to itself, and the alternative view, $\mathbf{B}\circ\mathbf{F}$, maps $\mathbb{R}^{MD}$ to itself.

# 4 Self-referential mechanisms in the past and in the future

We used the following standard self-referential schema in finite-dimensional DMMs. We required that $MN \leq D$ (this condition was unnecessary in the countably-dimensional case), and we used a row of $\mathbf{Y}$ to contain $\mathbf{W}$ to be used as the network matrix on the next linear step of the two-stroke cycle.

Now, as $\mathbf{F}$ produces not just $\mathbf{Y}$, but the pair $\langle \mathbf{W}, \mathbf{Y} \rangle$, we anticipate that a more considerable variety of self-referential mechanisms becomes available eventually. Embedding of $\mathbf{W}$ into $\mathbf{Y}$, which has been necessary for self-modification, is now optional.

# 5 Asymmetries between inputs and outputs

Considering these asymmetries will help us to generalize from the traditional view of $\mathbf{F}$ as being assembled from separate neurons.

We also hope that these asymmetries will help us to achieve a better way of handling our attempts to consider general *V-values* as network matrices[3].

In particular, a single row of $\mathbf{W}$ only produces one row of $\mathbf{X}$, but is using the whole $\mathbf{Y}$ as its input.

If we take a very traditional view of $\mathbf{F}$, then it would consist of single neurons mapping one row of $\mathbf{X}$ into one row of $\mathbf{Y}$. A bit more general (but still traditional for DMMs) consideration allows to associate with each neuron its own set of rows of $\mathbf{X}$ and its own set of rows $\mathbf{Y}$, and all these sets are non-intersecting (inputs and outputs are not shared between different neurons).

Emphasizing the asymmetry between inputs and outputs, the neurons of $\mathbf{F}$ (if we still insist of thinking about $\mathbf{F}$ as stratified into the neurons) should not be weaker than single rows of $\mathbf{W}$ in any respect. So we can allow them to have access to all $\mathbf{X}$ (this thought occurred to me during our earlier experiments with pure lightweight DMMs, but never made it into implementation).

At the same time, an element of $\mathbf{F}$ output (perhaps, a row of $\mathbf{Y}$ - let's think this way for simplicity for a second) can be generated by no more than one neuron in $\mathbf{F}$. If something in $\mathbf{Y}$ is not covered by a neuron, we can always add a placeholder `const` neuron to denote that this part of $\mathbf{Y}$ does not change.

The generalization we are about to consider is that it might not be necessary to stratify the output of $\mathbf{F}$ by rows (by neurons generating specific rows).

One still might need to stratify parts of the output of $\mathbf{F}$ by the method of change. E.g. we've seen earlier that when the network aggressively changes the whole $\mathbf{W}$ on each step, it is not so easy to handle training signals or manual update signals for $\mathbf{W}$ (such signals can be received and incorporated into $\Delta\mathbf{W}$, but $\mathbf{W}$ itself is very much a moving target).

## 6 Shader style for F

We need to produce a pair of matrices $\langle \mathbf{W}, \mathbf{Y} \rangle$ (or one can think about this as a single matrix $N \times (M+D)$, if more convenient). The natural style here might be shader-like style of $\mathbf{F}$, where the resulting real number (or a small-dimensional vector) is a function of coordinates in the output matrix.

This style tends to works nicely on graphic cards (which is why this is essentially the way *fragment shaders* (also known as *pixel shaders*) work). So this fits well with our goals of comfortably implementing DMMs on GPUs.

This style is often used in generative computer art[4].

It was observed earlier that the input-output structure of compositional pattern-producing networks is the same as the structure of fragment shaders[5]. This is quite suggestive for possible methods of DMM synthesis.

I am not sure how well this would work for sparse situation, which is typical for DMMs. Sparse textures do exist (e.g. in OpenGL the sparse textures extension exists since version 4.4 (2013)). But at the first glance, the patterns of their use are quite different from the patterns we need (which is, only compute the shader for a selected subset of target points). It should be not too difficult to express such patterns via multiplicative masks, or even with `if` statements and `discard` statements, but would the compiler optimize the results well?

## Notes

[1]For the state of DMMs as of November 2018, see **DMM technical report 11-2018**, "Dataflow matrix machines: recent experiments and notes for next steps", `https://github.com/jsa-aerial/DMM/tree/master/technical-report-2018`.

For the design efforts for the next generation of DMMs, see `https://github.com/anhinga/2019-design-notes`. For related preliminary code explorations, see `https://github.com/anhinga/2019-python-drafts`.

[2]See "Overcoming the vanishing gradient problem in plain recurrent networks" by Yuhuang Hu, Adrian Huber, Jithendar Anumula, Shih-Chii Liu, `https://arxiv.org/abs/1801.06105`; see also my note "Understanding Recurrent Identity Networks", `https://www.cs.brandeis.edu/~bukatin/recurrent-identity-networks.html` written about **v1** of that preprint, `https://arxiv.org/abs/1801.06105v1`.

[3]Section 2.3 of **DMM technical report 11-2018**, "Dataflow matrix machines: recent experiments and notes for next steps", `https://github.com/jsa-aerial/DMM/tree/master/technical-report-2018`. A particularly difficult part was trying to have an arbitrary nested structure of the indices of matrix rows, and to add the resulting outputs together. The present paper seems to be saying that it was misguided to try to do that; instead we should just have arbitrary nested structure of the indices of matrix columns (I am still thinking in terms of $\mathbf{F} \circ \mathbf{W}$ architecture here rather than $\mathbf{F} \circ \mathbf{B}$; what is really needed is to rethink this from the $\mathbf{F} \circ \mathbf{B}$ viewpoint).

[4]See e.g. "Random art in Python" by Andrej Bauer, `http://math.andrej.com/2010/04/21/random-art-in-python/` or "Random Art" by Christopher Stone, `http://nifty.stanford.edu/2009/stone-random-art/`

[5]For example, see "Interactive CPPNs in GLSL" by Xavier Snelgrove and Matthew Tesfaldet, `https://nips2018creativity.github.io/doc/interactive_cppns_in_glsl.pdf`, who use this to convert CPPNs to GLSL obtaining interactive neural shaders, `https://github.com/wxs/cppn-to-glsl`