# Bicomodules for Poly-shaped reservoir computer

André Muricy Santos

March 1, 2024

## 1 What are reservoir computers?

Reservoir computing is a machine learning technique for learning complex time series. A reservoir is an open, random, discrete dynamical system with many dimensions - called “nodes”. The reservoir system has input weights connecting to each node, internal weights connecting each node to each other node (though this architecture is not necessary - sometimes they’re arranged in a ring or other topologies), and output weights connecting each node to an output. The input and internal nodes are fixed forever, and the output weights are trained after enough training samples (where each training sample is one step of the input sequences). There’s three basic regimes/modes to most reservoir architectures. Here they are:

### 1.1 Training

The reservoir does a couple things. Each timestep, it takes an input sample (a vector of the dimension of the sequence we’re trying to predict - e.g. for the Lorenz system, this will be of length 3; the x-y-z coordinates) and multiplies it by its input weights (a matrix of dimensions `systemDim` $\times$ `amtOfNodes` ). It then takes this result and adds it to another matrix which is the result of multiplying the current states of every node by the internal weights. Finally this matrix goes through an activation function (`tanh` for example) and this becomes its new state! The input that led to this new state, along with the new state itself, is then stored and remembered in a list. This is repeated for a number of training steps, which leads the list to grow - it then looks like a ”history” matrix. After this is repeated enough times, the history matrix is used to produce an output weight matrix with linear regression. We then move to the next regime.

### 1.2 Warmup/touching

The reservoir, now trained, needs to get rid of so-called “transient dynamics”. The idea is that, after training, its states carry a lot of information about training dynamics (which were produced by some “training” input sequence). These need to be “washed away”. The reservoir is fed a bunch of training samples of the sequence it’s supposed to predict now, just to update its internal state according to these for a few timesteps (it doesn’t use its output weights yet). I also like calling this regime ”touching” and the next one ”going”, because it feels like cranking a machine for a while and then letting it go and do its own thing.

### 1.3 Prediction/going

Finally, the reservoir stops receiving inputs from the sequence it wants to predict and instead starts taking in *its own output*. This is also its prediction of what the next step of the test/predicted sequence would be, according to its perspective.

So we have a dynamical system with three different modes, where in each mode the different components of the system are taking in inputs from and providing outputs to different places. What does this remind you of?

## 2 It’s a morphism in Poly!

The entire reservoir, from training to prediction, is basically a big morphism in Poly of type $Sy^S \to p_{train} + p_{warmup} + p_{prediction}$. The three different regimes/stages are represented by different summands in the target of the morphism, and that’s pretty cool. I have Agda code from my master’s thesis for this in the repo linked at the bottom [1], and also a link to the thesis itself [2].

[1] https://github.com/amuricys/polynomial-functors

[2] https://odr.chalmers.se/server/api/core/bitstreams/774ba4a2-58b4-407f-aa6b-b0209b6c7d70/content

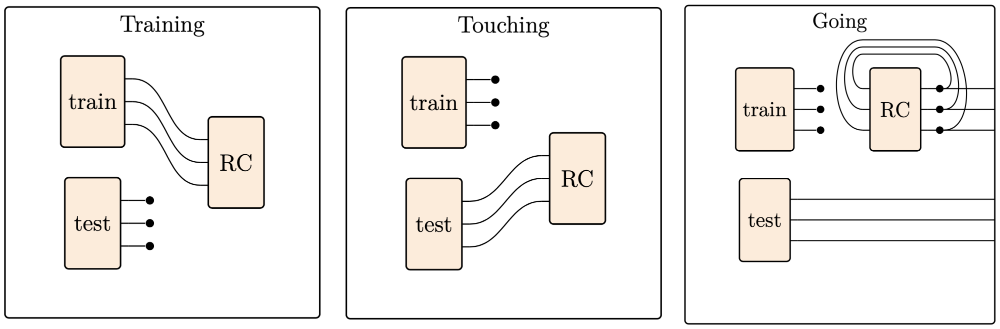

Figure 1: The three different regimes of a reservoir

There's just one problem: Morphisms of that shape could transition between modes arbitrarily. Simply knowing the type $Sy^S \to p + q$ doesn't tell you anything about the possibility of transitions between p-modes and q-modes. There's nothing preventing the implementation of the reservoir from going back from `pWarmup` to `pTrain`, for example. And this is where bicomodules come in.

## 2.1 Stronger mode-dependence

A morphism of bicomodules from this: $My \xleftarrow{Sy^S} Ny$ to this: $My \xleftarrow{p} Ny$ acts on a polynomial by "partitioning" it into $M$ polynomials on $N$-many variables. If we set $M = N$, it gives us what we want; we can think of the state space $S$ as $S_1 + S_2 + S_3 \ldots S_M$, where each $M$ corresponds to a mode. In our case, since we have 3 modes, we can think of our map $Sy^S \to p_{train} + p_{warmup} + p_{prediction}$ like this instead:

- $S_1 y^S \to p_{train}$ — the positions in $p_{train}$ are now only in y-variables that point to $p_{train}$ and $p_{warmup}$ - so $y_1$ and $y_2$
- $S_2 y^S \to p_{warmup}$ – the positions in $p_{warmup}$ are only in y-variables that point to $p_{warmup}$ and $p_{prediction}$ - so $y_2$ and $y_3$
- $S_3 y^S \to p_{prediction}$ – the positions in $p_{prediction}$ are only in y-variables that point to $p_{prediction}$ itself - so only $y_3$

The bicomodule data/laws force the directions in each of the partitioned polynomials to only lead to the subset of states that do the transitions that can be done. This is even more type-safe; a mode-dependent dynamical system can only perform transitions to regimes specified by the data in the bicomodule.

## 2.2 Future work

This is fine if we're thinking of a reservoir as only a single morphism, but ideally we would be able to split this further. There should be morphisms $S_{train} y^S_{train} \to p_{train}$, $S_{warmup} y^S_{warmup} \to p_{warmup}$ and $S_{pred} y^S_{pred} \to p_{pred}$ that correspond to each of these steps. The reason we want this is so that we can do fun things like comparing a morphism corresponding to a trained reservoir (the last one: $S_{pred} y^S_{pred}$ß$p_{pred}$) to a morphism corresponding to the sequence it predicts. **Chart**s in Poly can be used to compare systems, and the commuting squares of charts and lenses correspond to the fact that such lenses are really identical behaviorally. What if we had "approximately commuting" squares? Could these tell us anything about how to, for instance, optimally set hyperparameters (number of nodes, reservoir topology, distribution from which weights are drawn, activation function, number of training samples, etc)? The only problem is that these morphisms don't compose, and simply tensoring them immediately forces the resulting morphism to account for a mode-dependent system, which the input sequences are not. It's hard to make things type check.

I'm also thinking a lot about how to talk about bifurcations in dynamical systems in the language of Poly. I've had a sneaking intuition that the discreteness of bifurcations corresponds somehow to a change in wiring pattern in a mode-dependent system, and during the workshop I got to explore this a bit with Sophie, but it's still not clear where to go. Bicomodules allow us to partition the state space; could these partitions somehow correspond to different qualitative dynamics in a parametrized flow?

Anyway, this is very exciting stuff. I came to the workshop with all these ideas floating around my head, centering around the dialectic between continuous and discrete, how dynamical systems compute, how computation is possible at all etc. I'm leaving with even more questions, but also even more excited.