Putnamizing the Liquid State

Kevin Kirby
Northern Kentucky University

Echo state networks, liquid state machines, context reverberation networks– these all perform what has come to be known as "reservoir computing." This approach to neural computation has been getting much attention lately. I wish to show that not only is it of scientific and technological interest, but of philosophical interest as well. It is not so much that reservoir computing raises new philosophical problems, but that it casts a quarter-century old debate about how physical systems implement computations (arising from arguments made by Putnam and Searle) in a vivid new context.

I set the stage by doing a quick summary of reservoir computing. I then turn to Hilary Putnam's "Theorem" in an appendix to his *Representation and Reality*, and follow it as it is recast in the subsequent work of Chalmers, Scheutz and Joslin. I then consider the dynamical systems used in reservoir computing as a kind of prototype for the dynamics referred to by these philosophers. This leads to the general notion of a dynamical system interpreting another dynamical system. In reservoir computing we see the high dimensional dynamics of physical systems (even literally buckets of water) harnessed to create context for inputs to a connectionist network. This construction proceeds in a way analogous to Putnam's construction in the proof of his theorem. Perhaps a rock cannot simulate any finite state automaton, but, in some sense, a "reservoir" can!

Reservoir Computing

You are to build a neural network that learns to model a finite state automaton (FSA). More specifically, the neural network should learn to replicate a mapping of input sequences to output sequences determined by a given FSA. You want to do this in the supervised learning paradigm: a training set consisting of a table of (*input sequence*, *output sequence*) pairs is presented to the network repeatedly, and, over time, the network modifies its connection weights so that –you hope– it implements the finite mapping given in the training set. Moreover, you hope it *generalizes*, meaning that if you feed an arbitrary input sequence both to the network and to the finite automaton, they will produce the same output sequence. Since the output at any given time is determined not just by one input but by an undetermined amount of the entire input history, the neural network will need to maintain an internal state. Thus its output at a given time is a function of both the current input and its current state.

This is hard. To maintain an internal state in a connectionist network (of the usual simple model neurons) there must be recurrent connections. Algorithms to train feedforward networks can be adapted to recurrent networks, but fare poorly. And the idea of *learning internal states from temporal experience* seems intrinsically difficult anyway. After all, what is a "good" state space for a problem? And how can we get there by tweaking synaptic connections?

One path to the solution is to just give up on the hardest part: let's not even try to modify the recurrent connections. Let's just imagine we have a high dimensional dynamical system which happens (for the moment) to be a totally connected network of simple threshold neurons. The connection weights in this recurrent subnetwork are fixed and random. External inputs impinge on this subnetwork. A simple feedforward network (perhaps just a single layer) also receives these external inputs as well as outputs from the recurrent subnetwork. Being a feedforward network, it can be trained with very simple algorithms (such as the perceptron learning rule or its variants), and thus learns associations between the current input and the state of the recurrent network. See Figure 1, left.

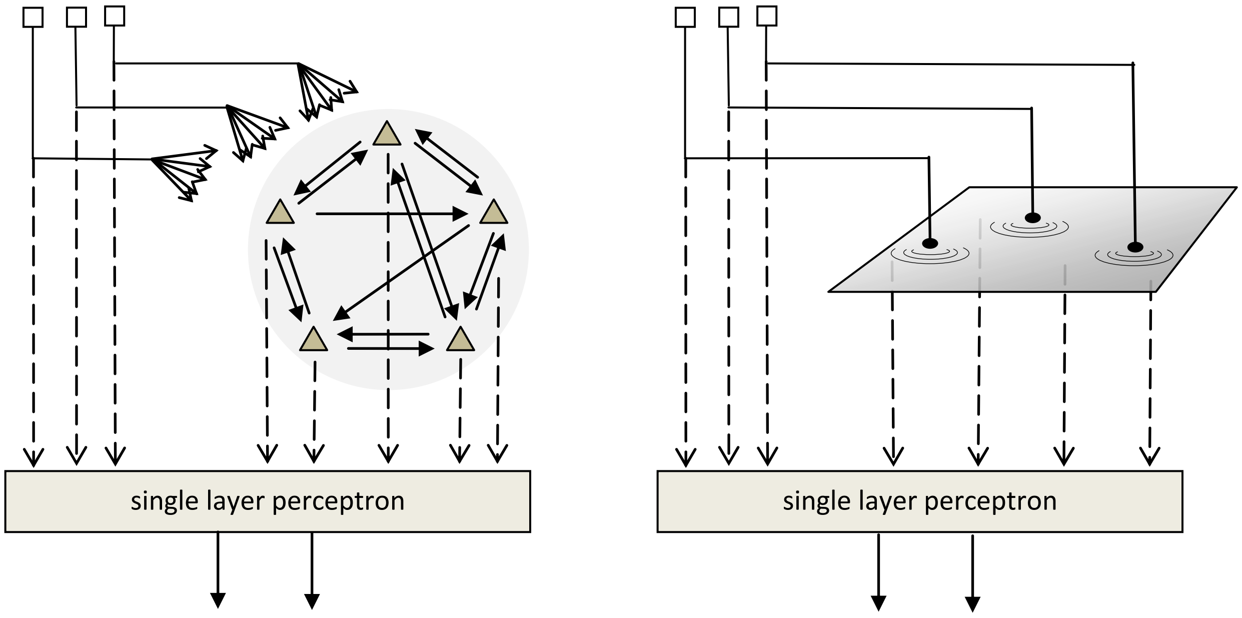

Figure 1. A neural network with fixed random connections (*left*) and a reaction-diffusion system (*right*) are two early examples of reservoir computation. They both hold compressed representations of input histories, allowing a trainable layer of neurons (e.g. a classical perceptron) to learn associations between current inputs (top layer) and accumulated context that reverberates around the reservoir. Fixed connections are shown as solid; trainable connections are dashed.

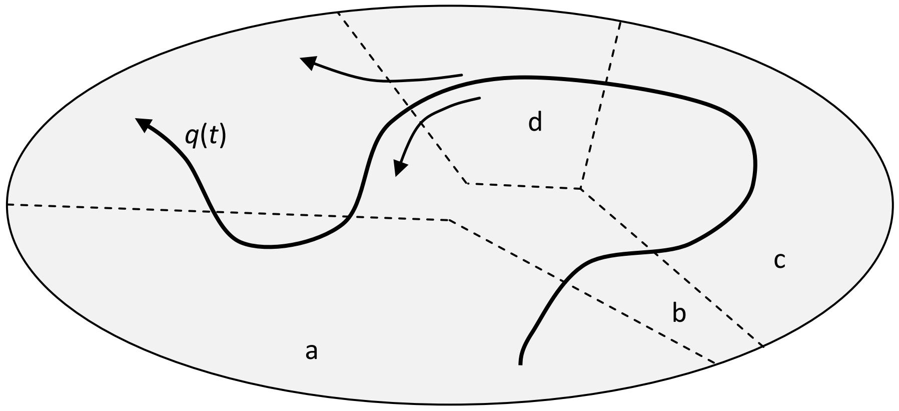

Figure 2. The trajectory of a physical system through regions of a state space arbitrarily mapped to states of a given finite automaton, in the construction of Putnam (1988). This yields the discrete sequence of states *abcdbab*. This is analogous to the trajectory through a high dimensional state space in a reservoir. See Rabinovich (2008) for a similar figure drawn from the neurobiological literature. Once inputs are introduced, the trajectory can be forced swerve into different regions of the space. In reservoir computing these states are interpreted by an output layer, which adaptively carves up the state space into potentially meaningful regions, which can be called "context."

This architecture was apparently first explored by Gallant and King (1988). They were making a new attempt to use fixed random connections in neural learning, an idea present in the earliest neural network investigations of Rosenblatt (1961). They called their totally connected subnetwork a "Sequential Associative Memory" (SAM). They experimented on some simple problems found the contemporary literature, with some success. For example, they could learn the sequential parity task (where the output is to be 1 iff the number of 1-bits received is odd) with only 10 "SAM cells".

Kirby and Day (1990) noted that the deep idea here was not in the details of the totally connected neural network, but that it served as a high-dimensional fixed "repository of context." Inputs impinging on the fixed random subnetwork set and reset the internal dynamics of the system, causing it to swerve into different paths in a high dimensional state space. The output layer adaptively sorted out this "encrypted" input history. For this partial representation of input history they used the term *context*, and named the overall architecture a *Context Reverberation Network*. They showed through simulations that even nearest neighbor connectivity in the subnetwork would suffice, albeit with longer learning times. Generalizing further, they let the context reverberation subsystem become a continuous dynamical system: a reaction-diffusion system based on the morphogenesis model of Turing (1952). See Figure 1, *right*. Kirby (1991a, 1991b) subsequently explored connections between dynamical systems as repositories of context and sequential learning and simulation of finite automata.

At present there is a surge of interest in this style of computing, stimulated by their subsequent re-introduction as *Echo-State Networks* by Jaeger (2001) and as *Liquid State Machines* by Maas et al (2002), each with slightly different emphases. This has led to many experimental results as well as to a clear theoretical foundation which ties it to dynamical systems theory. The idea of using high dimensional dynamical subsystems to help process temporal information has even been useful in the study of biological neural networks, from the cerebellum to olfactory systems (see, for example, Rabinovitch et al 2008).

The survey by Schrauwen et al (2007) uses the term *reservoir computing* for such approaches. That is the term I will use here. A reservoir of what? Following Kirby and Day, I would say: *the context for the current input*. The network produces the appropriate output "in that context."

Interestingly, the reservoir idea can be implemented with amazing literalness. Fernando and Sojakka (2003) used an actual bucket of water to serve as the repository of context. They set a transparent water tank on an overhead projector and used weights to create eight point sources of waves and read out the interference patterns as projected onto a 2D image.

Putnam and the Implementation Relation

Using the states of a bucket of water to do computation may remind some philosophers of "Hinckfuss's Pail" as described by Lycan (1987). Here the high dimensional dynamics (the "seething complexity") of a pail of seawater could allow someone (say a computationalism-friendly philosopher) to imagine a mapping from the states of water to the states of a mind. It is this kind of imagined mapping I wish to explore.

At the end of *Representation and Reality*, Putnam (1988) argues for a proposition (which he calls a Theorem) that runs as follows: *Every ordinary open system is a realization of every abstract finite automaton.* This proposition played a role in that book's repudiation of functionalism, by leading to the conclusion that, in short, "functionalism implies behaviorism." This argument has become fairly well known, so I only give a quick and rough overview here.

Putnam first treated the case of an autonomous dynamical system, i.e. one with no inputs or outputs. The physical state $q(t)$ was a continuous function of time. His idea was that you could map this state space into the finite state space of an FSA in such a way that the image of $q(t)$ under this mapping led to any desired sequence of automaton states. This meant that dynamical system *implemented* the FSA. He then adapted this to the case of inputs and outputs. The key move was that one could *interpret* the configuration as being in certain FSA states. In other words, an externally forced physical system was open to unconstrained interpretation as a discrete computing device. See Figure 2. A similar but less formal argument allowed Searle (1992) to imagine his wall was executing a word processing program.

Chalmers (1996) repaired some technical deficiencies in Putnam's argument, and showed that it pointed to the need for a more refined notion of implementation. Stating that "causal relations among physical states must mirror formal relations among automaton states," his extended construction had a few fixes (a "clock" to guarantee nonperiodicity, a "dial" to make sure all physical states were in the mapping, and a "video camera" to account for input history). He finally broke up "monadic" states into products of states of smaller components, $q(t)= (q_1(t), q_2(t), \ldots, q_n(t))$, as a step for capturing spatially distinct components of a device to which we can realistically impute computing.

But in the end the challenge here is that the notion of *mapping* is quite odd. Mathematically we can easily think of a mapping between sets, but it is quite a different and difficult thing to think of a mapping between *a piece of the physical world* and a set. Scheutz (1999) addressed this by lifting the mapping so that it applied between formal domains- a mathematically formulated physical theory (e.g. electric circuits) and the formal dynamics of the given automaton. Joslin (2006) moved further towards a convincing theory of implementation (or rather, "realization" of a computation by a physical system) by turning the mapping around. With Putnam and Chalmers, it was as if one started with a particular FSA and arbitrarily mapped its states onto a physical system. According to Joslin, it is more relevant to start with a real physical system and ask what kind of computing it can be seen as doing well. Thus, humans seeking computing in nature do not arbitrarily construct interpretations of physical states. Interpretation is guided by what physical features seem *salient* for computing.

In the end, this connects to the idea of modeling. What a state of physical system is is theory-dependent. A piece of the physical world is not a *system* until we assign states to it in a theory. (I could put a ball on a string in front of five physicists and ask them to give me the Lagrangian function for the system, and could get five different answers.) The art of coming up with a good state space is the art of modeling. Suppose we are looking at a lake whose fish population is fluctuating. Like Joslin in the realization problem, we go in the direction of moving from the real world to a formal system, say a differential equation for predator-prey dynamics. We don't go in the other direction, from a differential equation to a piece of the world it of which it is a model.

Or do we? Think of building a computer. We harness the resources of nature for computing. We do seek pieces of the world that can function as memory units, logic gates, and so on. In the days of analog computing we sought out physical devices whose resistance, capacitance and inductance behaved in such a way that they were modeled by linear differential equations. (This is the direction Scheutz emphasizes.) So we go in both directions: we *model* pieces of the world with computational formalisms, and we *implement* computational formalisms with pieces of the world. And, inevitably, there are better and worse models, better and worse implementations.

In one frame of mind we can regard physical systems as if they were Rorschach blots that could be interpreted arbitrarily. Looking at Hinckfuss's pail we can imagine a kind of Hermeneutic Demon telling us that it is implementing a program for computing, say, hyperbolic tangents, showing us how the state transitions line up. (To *putnamize*: to arbitrarily map the states of a dynamical system to yield another system of one's choice.) Perhaps we should view *Computes* as a three-place relation connecting formal systems, physical systems, and an interpreter. A demonic interpreter inserted into the third slot of *Computes*(f,p,i) might yield a true sentence, but perhaps not an informative one. The admissibility of such interpretations is the source of "triviality arguments" against functionalism (Godfrey-Smith 2008).

Interpreting Reservoir Dynamics

Recall the overall purpose for using a reservoir computing system: *to learn to model a finite state automaton*. But wait: is *model* the right word? Should we not say the reservoir computing system must learn to *implement* a given FSA? After all, our reservoir computing system may contain a cerebellum or a bucket of water, and is thus a physical system trying to track a formal system. But then consider this one more time: the FSA might be a formal system (the experimenters might write down a regular expression they want the system to learn), but, on the other hand, the input/output strings might very well come from the real-world environment. Many neural network experiments have drawn input patterns in real time from the external world, e.g. cameras on the front of a moving truck.

So in fact, *implementation* and *modeling* are special cases of a general relation between two systems. This general relation, I would argue, is the interpretation of one system by another. And I claim this is pervasive. Pervasive in the brain, perhaps pervasive outside the head as well.

To explore this, let us peer into a reservoir. Take the random neural network or the reaction-diffusion system in Figure 1, or even the bucket of water in the experiments of Fernando and Sojakka. In any case, a stream of inputs has been impinging onto the reservoir for some time, setting up changing patterns. Putting this in a state space, we see a trajectory moving through a high dimensional space, as in Figure 2. The fact that the state, call it $q(t)$, is in a very high dimensional space, means it has *way too much information* to be useful. The readout layer collapses this high dimensional state space into a smaller set of output patterns, depending on the current input. This follows Putnam's original construction. Instead of asking the network to implement a function $y(t)= F(x(t), x(t–1), \dots x(t–n))$, the reservoir creates a state $\mathrm{q}(t)$ that captures the salient features of the inputs $x(t)$, $\mathrm{x}(t–1), \dots \mathrm{x}(t–n)$, so that instead we ask it merely to implement a function $F(x(t), \mathrm{q}(t))$.

Now, this mapping is not engineered from the beginning. The idea in context reverberation learning is to employ a simple associationist learning algorithm (the bottom layer in Figure 1) to make the associations between the context and the current input. As training proceeds, we are judging on purely *behavioral* terms: do input sequences map to the correct output sequences? At first no, then, later, we hope, yes. But behind these behavioral changes is a system that is learning how to map its states into states that are meaningful for the sequential behavior it is trying to learn.

The state space is adaptively reduced by the needs of the problem at hand, as determined by the training set. Each region of the state space (e.g. a,b,c,d in Figure 2) collapses all the states that are relevant to processing the inputs; they are the context in which the current input produces an output. Earlier approaches to training recurrent neural networks involved modifying the recurrent connections themselves. This would mean that input streams would be constantly "recontextualized" as learning proceeded. Without a fixed framework for assigning meaningful pieces of the state space, it seems inevitable learning would be very difficult, as it in fact proved to be. The fact that the dynamics here can be continuous means that we are now outside the realm of "mechanism" and thus outside the realm of computation, according to the classification of Piccinini (2008).

Pulling back from the details, in reservoir computing the field of neural networks has hit upon the importance of the interpretation of dynamical systems by dynamical systems. If you run a training algorithm on a reservoir computing system twice, it may carve up the state space differently each time. There is indeed some arbitrariness in the mapping. But there is also a constraint: the state space has to serve the problem at hand. In this case, there is no room for a hermeneutic demon.

References

Chalmers, D.J. (1996). Does a rock implement every finite-state automaton? *Synthese,* 108, 310–333.

Fernando, C., & Sojakka, S. (2003). Pattern recognition in a bucket. In Banzhaf, W., Christaller, T., Dittrich, P., Kim, J.T. & Ziegler, J. (Eds.) *Advances in Artificial Life*: *Proceedings of the 7th European Conference on Artificial Life, ECAL 2003*. Springer.

Fresco, N. (2008) An analysis of the criteria for evaluating adequate theories of computation. *Minds and Machines* 18:379-401.

Gallant, S.I., & King D.J. (1988). Experiments with Sequential Associative Memories. *Program of the 10th annual conference of the Cognitive Science Society* (pp.40-47). Hillsdale, NJ: Lawrence Erlbaum Associates.

Godfrey-Smith, P. (2008). Triviality arguments against functionalism. Philosophical Studies

Jaeger, H. (2001). The "echo state" approach to analyzing and training recurrent neural networks. *Technical Report 154, German National Research Center for Information Technology*.

Joslin, D. (2006). Real realization: Dennett's real patterns versus Putnam's ubiquitous automata. *Mind and Machines* 16, 29–41.

Kirby, K.G., & Day, N. (1990). The neurodynamics of context reverberation learning. *Proceedings of the Annual Conference of the IEEE Engineering in Medicine and Biology Society* 12 (4), 1781–1782.

Kirby, K.G. (1991a). Context dynamics in neural sequential learning. *Proceedings of the Florida Artificial Intelligence Research Symposium* (*FLAIRS 1991*), 66-70.

Kirby, K.G. (1991b). Duality in neurocomputational inductive inference: a simulationist perspective. *Proceedings of the IEEE International Conference on Systems Engineering*, 351-354.

Lycan, W.G. (1987). *Consciousness*. MIT Press.

Maas, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation* 14(11): 2531–2560.

Piccinini, G. (2008) Some neural networks compute, others don't. *Neural Networks*.

Putnam, H. (1988) *Representation and reality*. Cambridge, Massachusetts: MIT Press.

Rabinovich, M., Huerta, R., & Laurent, G. (2008). Transient dynamics for neural processing. *Science* 321, 48–50.

Rosenblatt, F. (1961) *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Washington, DC: Spartan Press.

Scheutz, M. (1999). When physical systems realize functions. *Minds and Machines,* 9(2): 161–196.

Schrauwen, B., Verstrete, D., & Campenhout, J.V. (2007). An overview of reservoir computing: theory, applications, and implementations. *Proceedings of the European Symposium on Artificial Neural Networks ESANN 2007*, pp. 471-482.

Searle, J. (1992). *The rediscovery of the mind*. Cambridge, Massachusetts: MIT Press.

Turing, A.M. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society*, B237, 37–72.