DOI: 10.1103/PhysRevLett.120.024102

He, Pathak and their colleagues Brian Hunt, Michelle Girvan and Zhixin Lu (who is now at the University of Pennsylvania) achieved their results by synthesizing existing tools. Six or seven years ago, when the powerful algorithm known as "deep learning" was starting to master AI tasks like image and speech recognition, they started reading up on machine learning and thinking of clever ways to apply it to chaos. They learned of a handful of promising results predating the deep-learning revolution. Most importantly, in the early 2000s, Jaeger and fellow German chaos theorist Harald Haas made use of a network of randomly connected artificial neurons — which form the "reservoir" in reservoir computing — to learn the dynamics of three chaotically coevolving variables. After training on the three series of numbers, the network could predict the future values of the three variables out to an impressively distant horizon. However, when there were more than a few interacting variables, the computations became impossibly unwieldy. Ott and his colleagues needed a more efficient scheme to make reservoir computing relevant for large chaotic systems, which have huge numbers of interrelated variables. Every position along the front of an advancing flame, for example, has velocity components in three spatial directions to keep track of.

It took years to strike upon the straightforward solution. "What we exploited was the locality of the interactions" in spatially extended chaotic systems, Pathak said. Locality means variables in one place are influenced by variables at nearby places but not by places far away. "By using that," Pathak explained, "we can essentially break up the problem into chunks." That is, you can parallelize the problem, using one reservoir of neurons to learn about one patch of a system, another reservoir to learn about the next patch, and so on, with slight overlaps of neighboring domains to account for their interactions.

Parallelization allows the reservoir computing approach to handle chaotic systems of almost any size, as long as proportionate computer resources are dedicated to the task.

Ott explained reservoir computing as a three-step procedure. Say you want to use it to predict the evolution of a spreading fire. First, you measure the height of the flame at five different points along the flame front, continuing to measure the height at these points on the front as the flickering flame advances over a period of time. You feed these data-streams in to randomly chosen artificial neurons in the reservoir. The input data triggers the neurons to fire, triggering connected neurons in turn and sending a cascade of signals throughout the network.

The second step is to make the neural network learn the dynamics of the evolving flame front from the input data. To do this, as you feed data in, you also monitor the signal strengths of several randomly chosen neurons in the reservoir. Weighting and combining these signals in five different ways produces five numbers as outputs. The goal is to adjust the weights of the various signals that go into calculating the outputs until those outputs consistently match the next set of inputs — the five new heights measured a moment later along the flame front. "What you want is that the output should be the input at a slightly later time," Ott explained.

To learn the correct weights, the algorithm simply compares each set of outputs, or predicted flame heights at each of the five points, to the next set of inputs, or actual flame heights, increasing or decreasing the weights of the various signals each time in whichever way would have made their combinations give the correct values for the five outputs. From one time-step to the next, as the weights are tuned, the predictions gradually improve, until the algorithm is consistently able to predict the flame's state one time-step later.