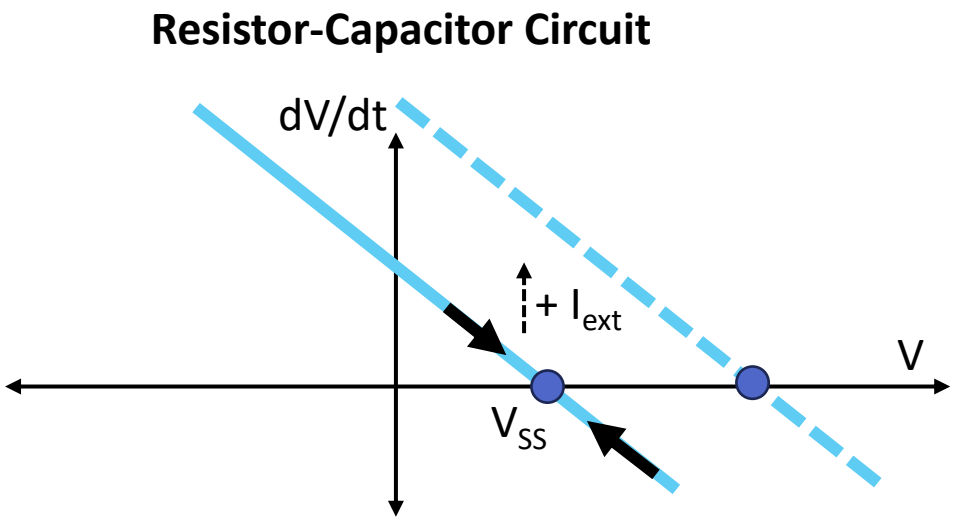
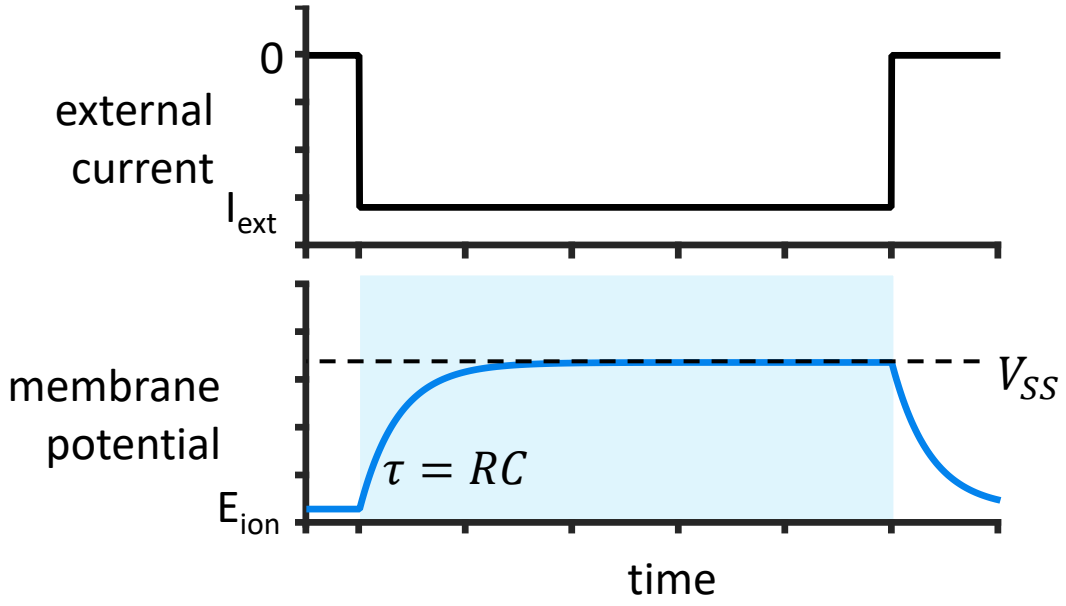
Computing with neural populations

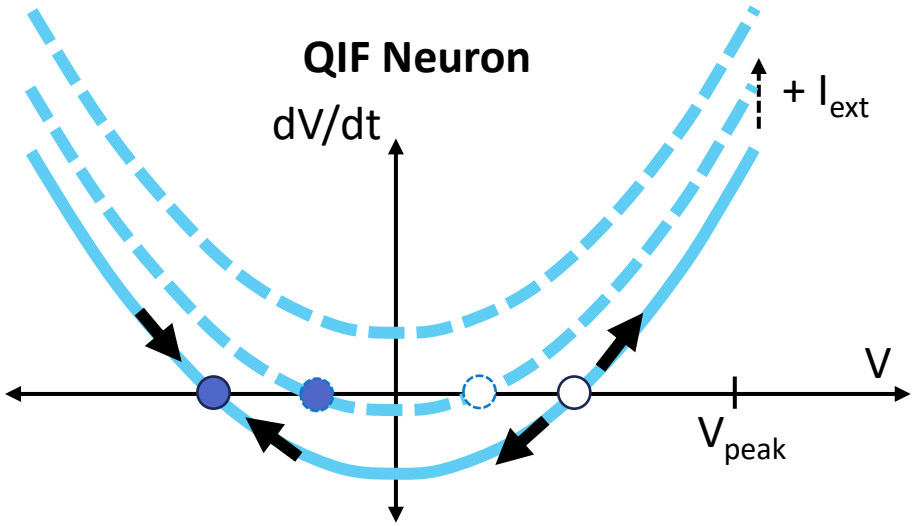
NUIN 443

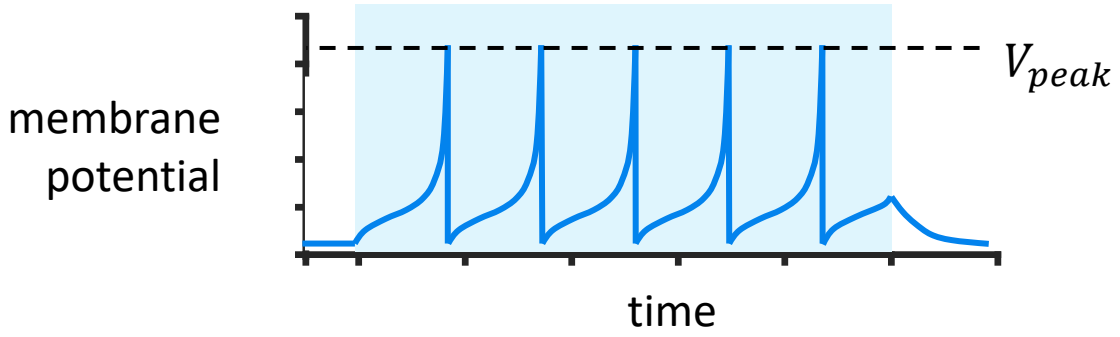
May 20, 2024

The story so far: 1D systems with fixed points and bifurcations



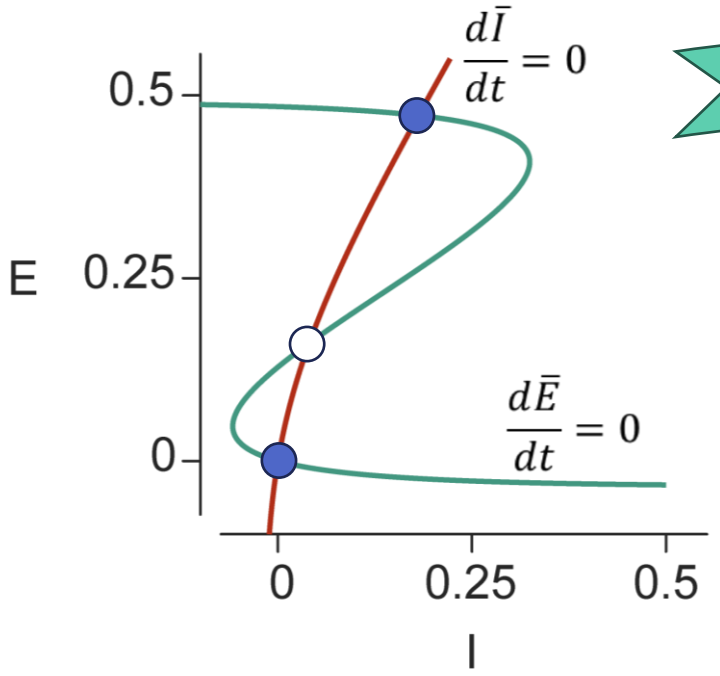




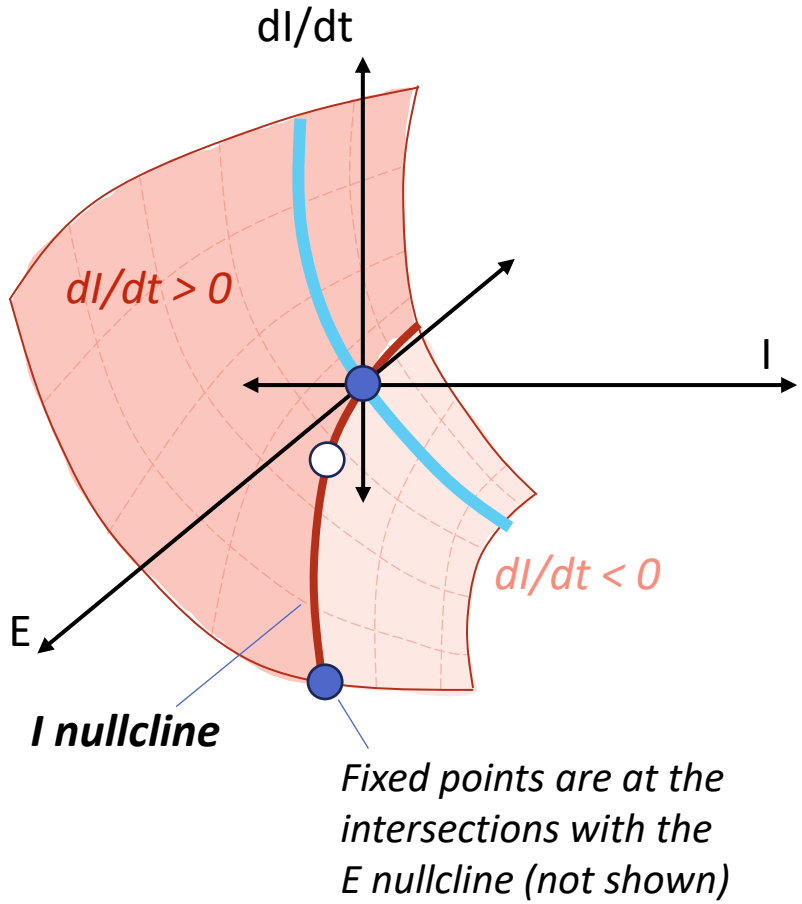


The story so far: 2D systems add nullclines and limit cycles

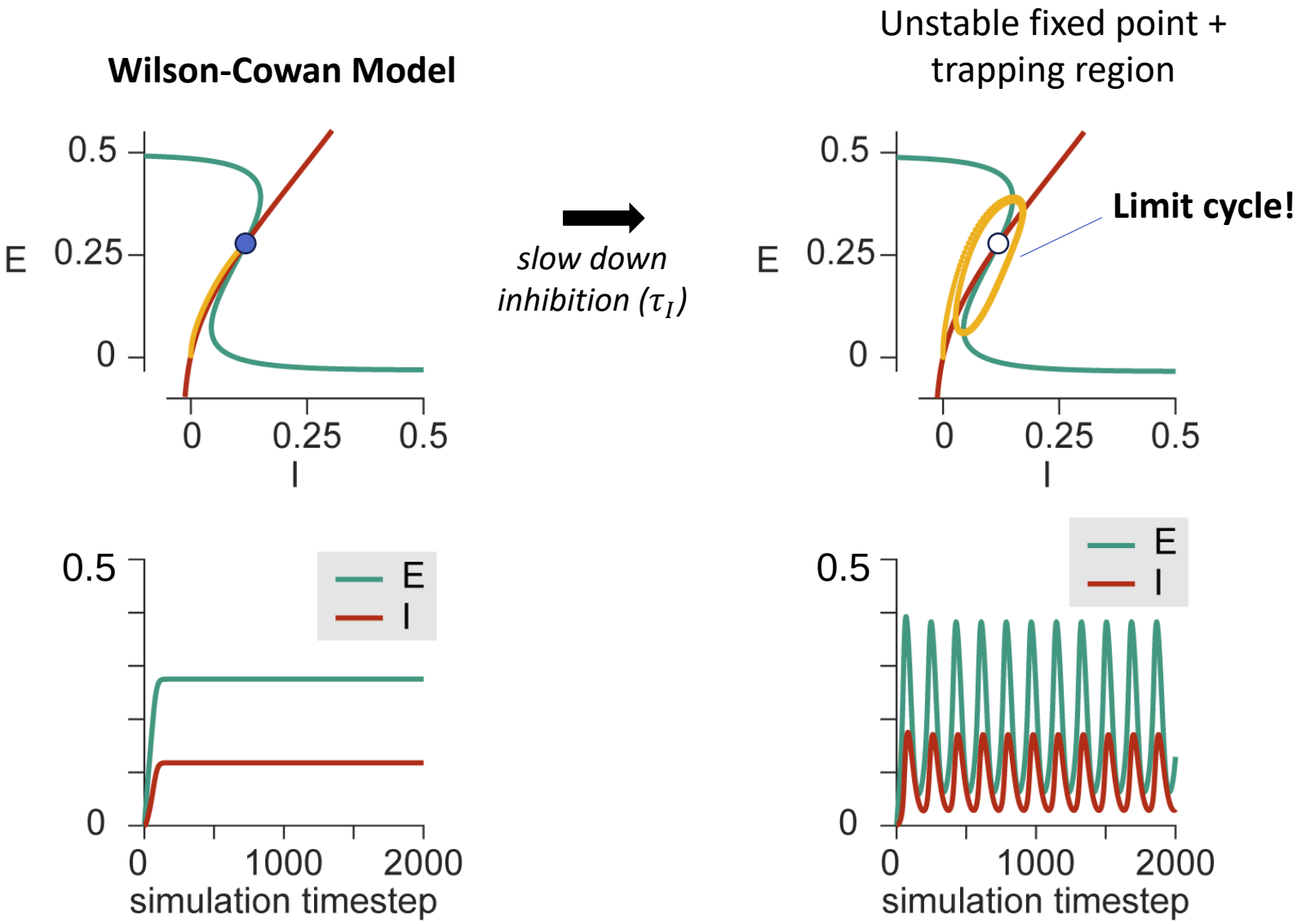
Wilson-Cowan Model



Nullclines!



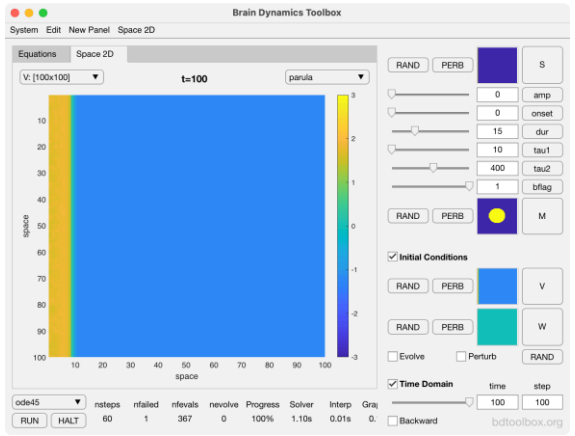
The story so far: 2D systems add nullclines and limit cycles

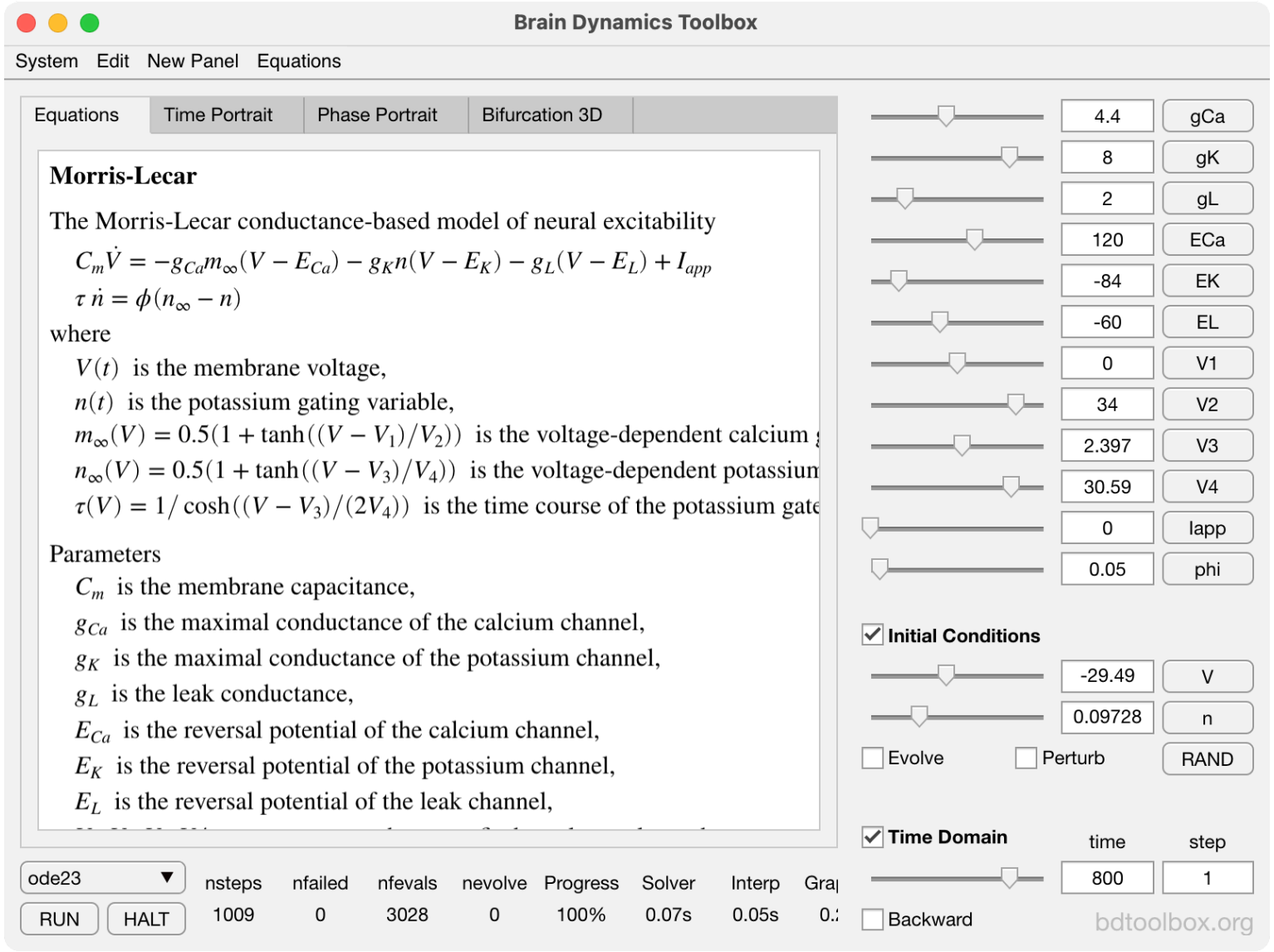


A (Matlab) tool to develop your intuition for these models

The Brain Dynamics Toolbox
Bdtoolbox.org

Interact with and simulate:
- Hodgkin-Huxley
- Morris-Lecar
- Hindmarsh-Rose (bursting cell)
- FitzHugh-Nagumo
- Wilson-Cowan
- 2D neural sheets





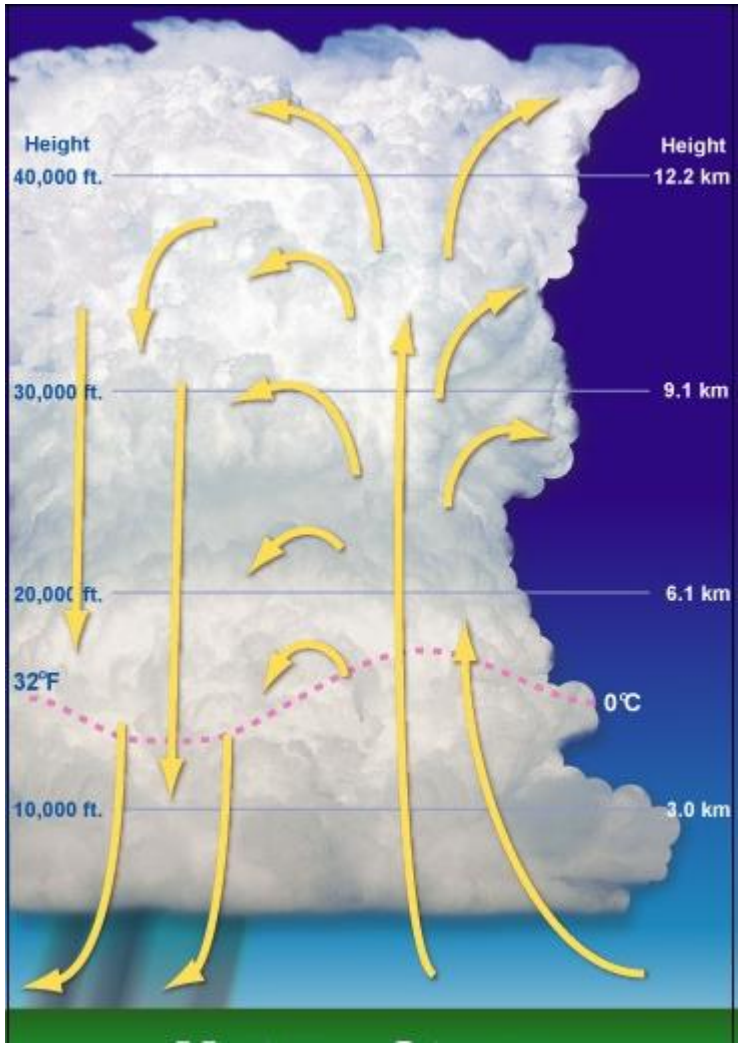
Moving into 3 or more dimension introduces a last potential solution type: **chaos**

In 1963, Edward Lorenz developed a 3D model of atmospheric convection, relating the rate of convection (x) to the horizontal (y) and vertical (z) temperature variation of the atmosphere, via a set of physical constants σ, ρ, and β.

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = x(\rho - z) - y$$

$$\frac{dz}{dt} = xy - \beta z$$



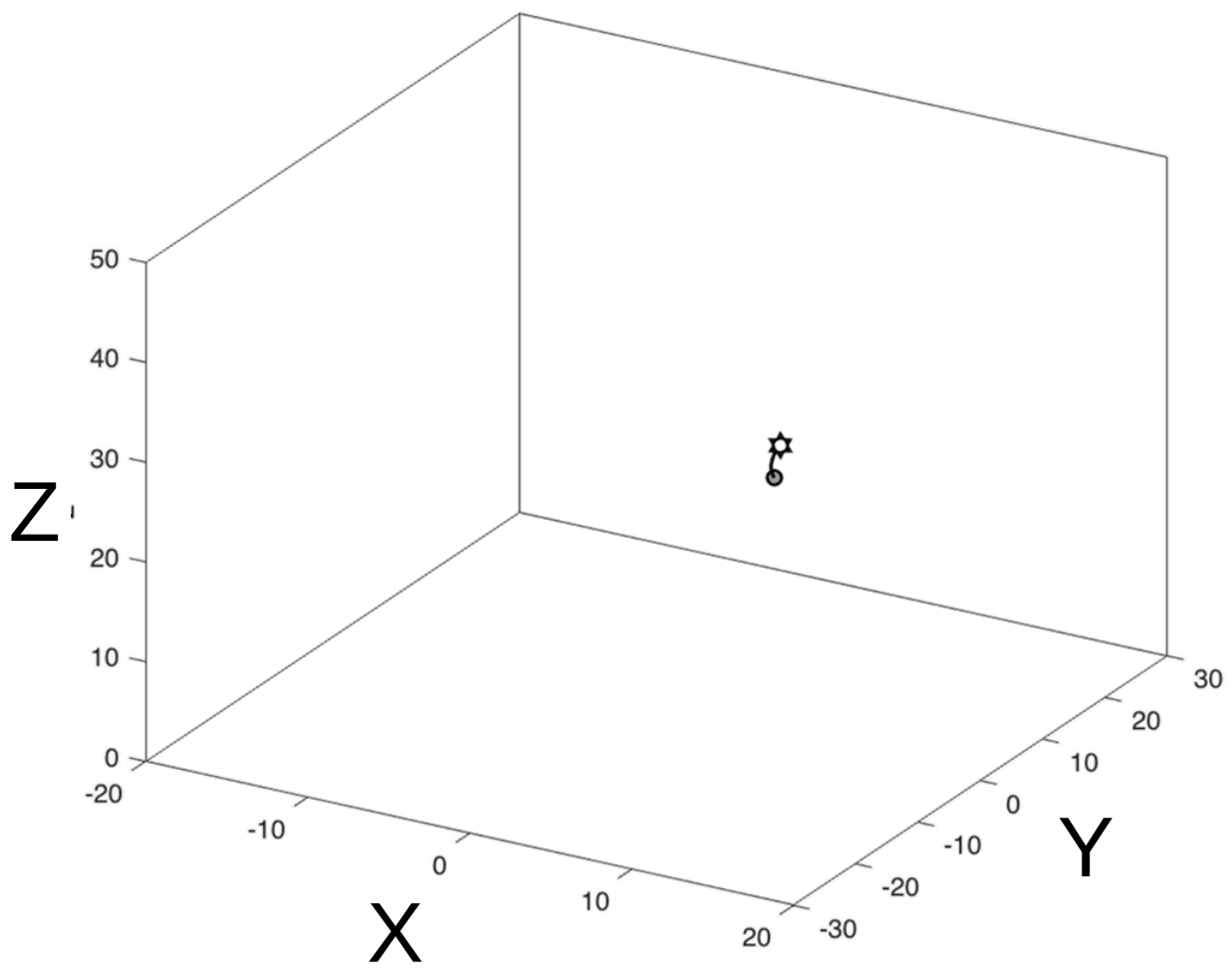
Moving into 3 or more dimension introduces a last potential solution type: **chaos**

In 1963, Edward Lorenz developed a 3D model of atmospheric convection, relating the rate of convection (x) to the horizontal (y) and vertical (z) temperature variation of the atmosphere, via a set of physical constants σ, ρ, and β.

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = x(\rho - z) - y$$

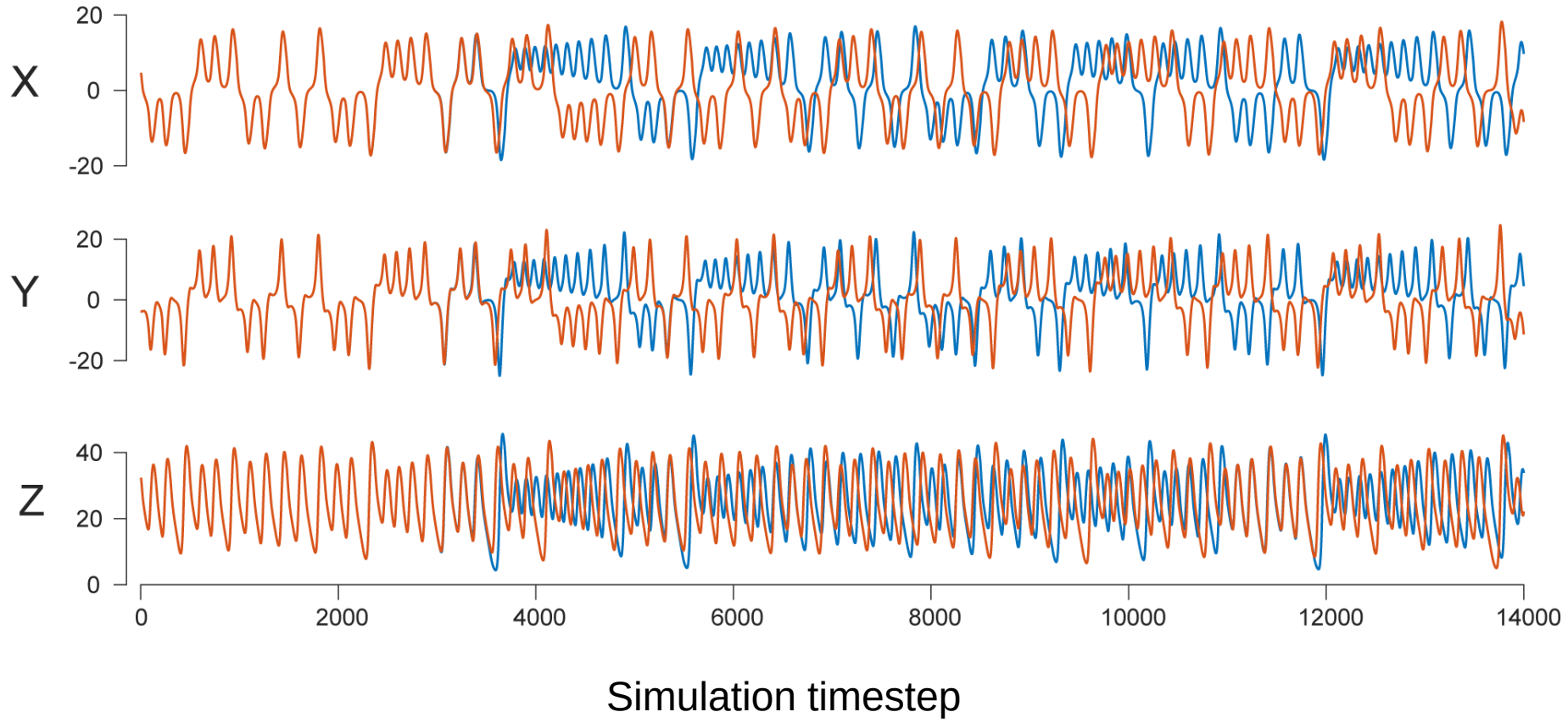
$$\frac{dz}{dt} = xy - \beta z$$



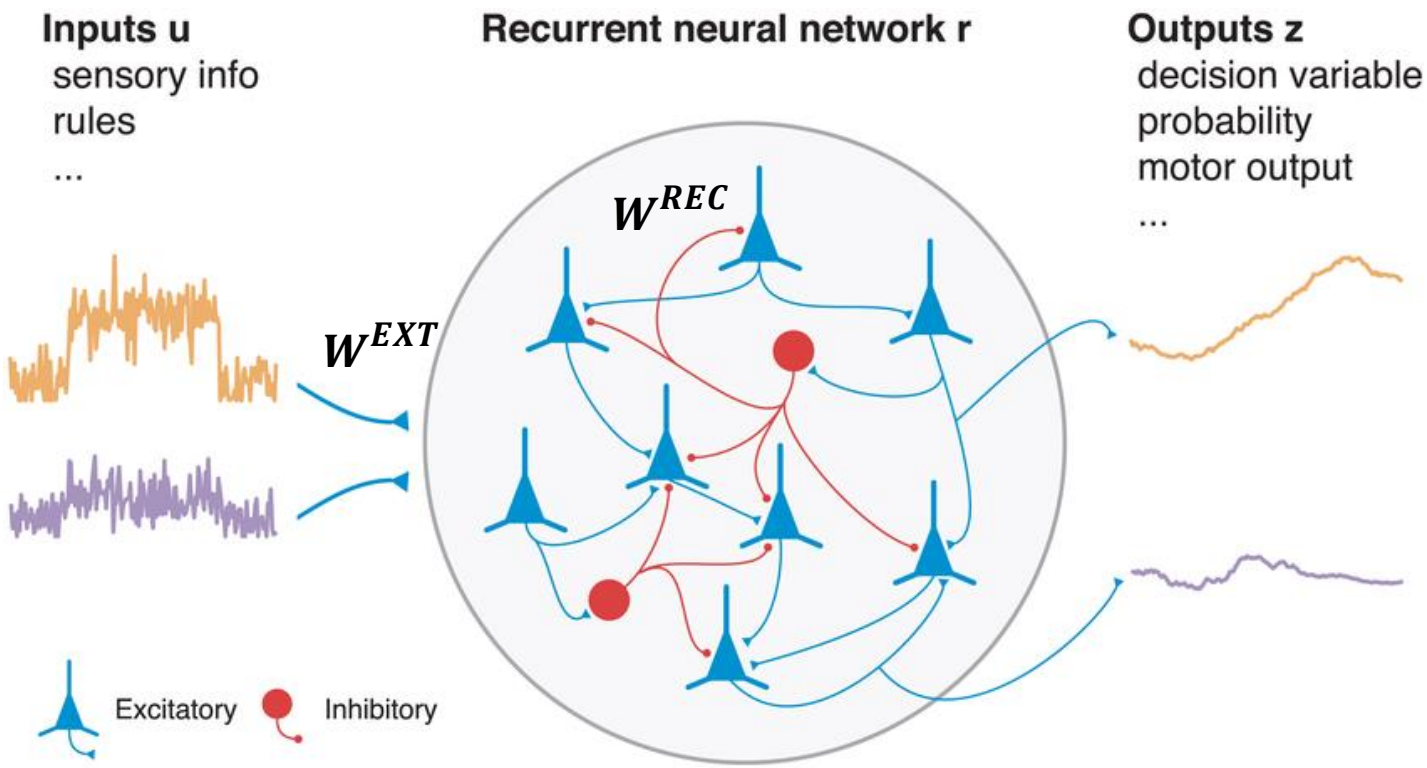
In a chaotic system, small differences in initial conditions are amplified with time

$$\begin{bmatrix} X(0) \\ Y(0) \\ Z(0) \end{bmatrix} = \begin{bmatrix} 4.661224 \\ -3.669068 \\ 32.355960 \end{bmatrix}$$

$$\begin{bmatrix} X(0) \\ Y(0) \\ Z(0) \end{bmatrix} = \begin{bmatrix} 4.661224 \\ -3.669068 \\ 32.355961 \end{bmatrix}$$



High-dimensional dynamical systems in neuroscience: recurrent neural networks (RNNs)



$$\tau\dot{x} = -x + W^{REC}r + W^{EXT}u$$

membrane potential — firing rate — external input

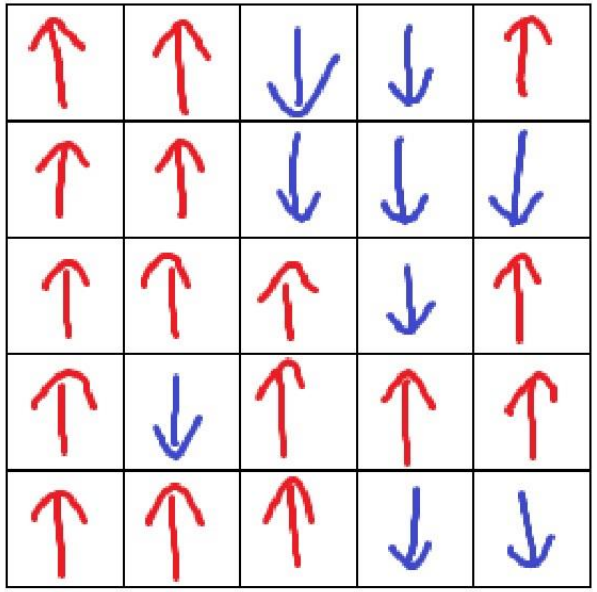
Song, Yang, and Wang 2016

The first “RNN”

The **Ising model** (1925) simulates interactions between “spins” of atomic nuclei arranged in space.

Each particle has a + or – spin, and configurations where neighboring particles have the same spin are energetically favorable.

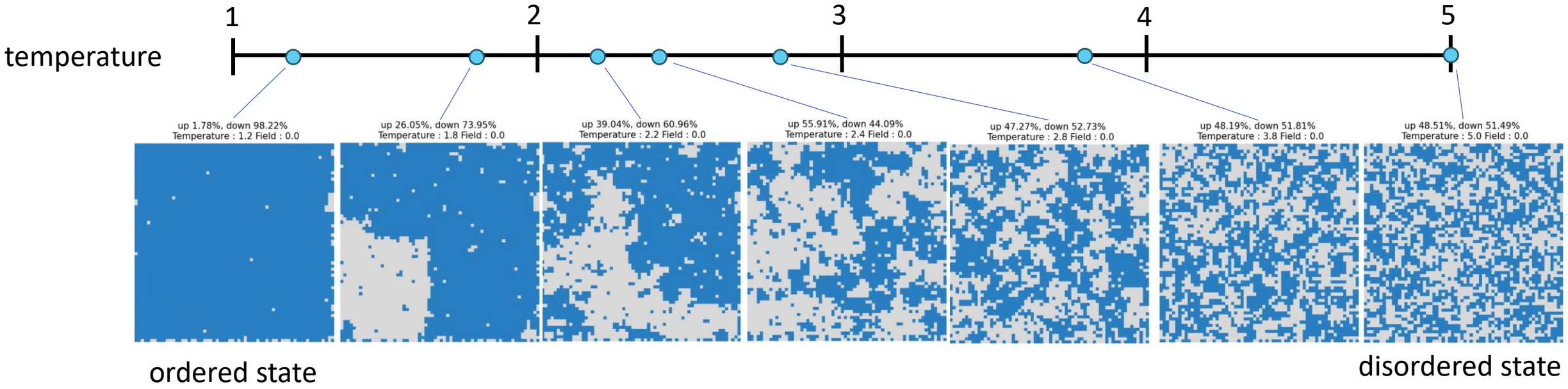
2D grid form (solved 1943)



$$H(\sigma) = -\sum_{i,j} J_{ij}\sigma_i\sigma_j$$

$$P_\beta(\sigma) = \frac{\exp(-\beta H(\sigma))}{Z_\beta}$$

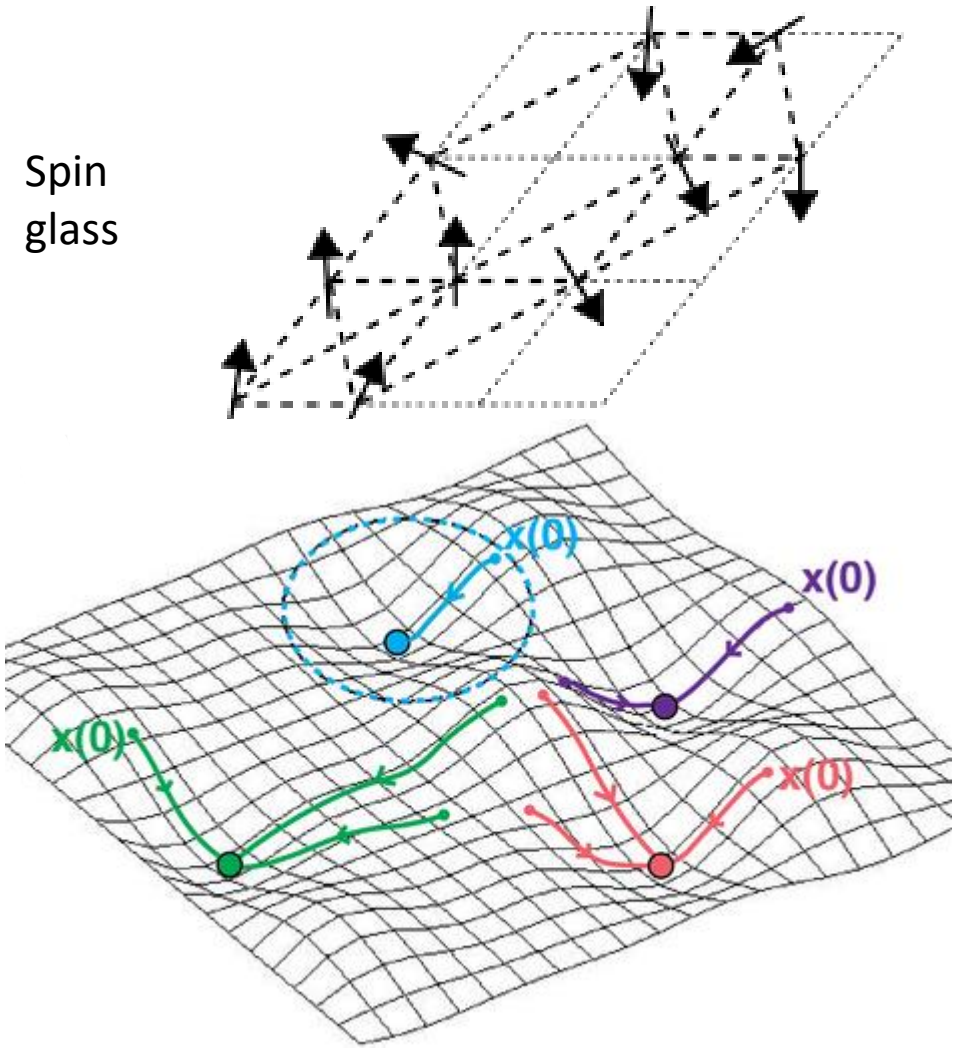
Where β is temperature and J_{ij} is the **strength of interaction** between particles i and j.



https://www.youtube.com/watch?v=eKmDtG3D2Wc

From Ising models to spin glasses

Spin glass



In 1975, Edwards and Anderson introduced two generalizations of the Ising model:

- Allow the spin to be an arbitrary 2d vector instead of +/- 1
- Allow the coupling matrix J_{ij} between particles to be non-uniform
  - Specifically, assume the distribution of couplings between particle pairs to be Gaussian

$$H(\sigma) = -\sum_{i,j} J_{ij} S_i S$$

$$P(J_{ij}) \sim \mathcal{N}(J_0, J^2)$$

Studying this system in 1975, Sherrington and Kirkpatrick hypothesize that such a model could have many local minima (ie stable, attracting states).

What if we could adjust the J_{ij}'s of our spin glass?

In 1972, Shun-ichi Amari proposed a way to **set the values of J_{ij}** to so that the system could store and recall patterns and sequences as stable equilibrium states of the system.

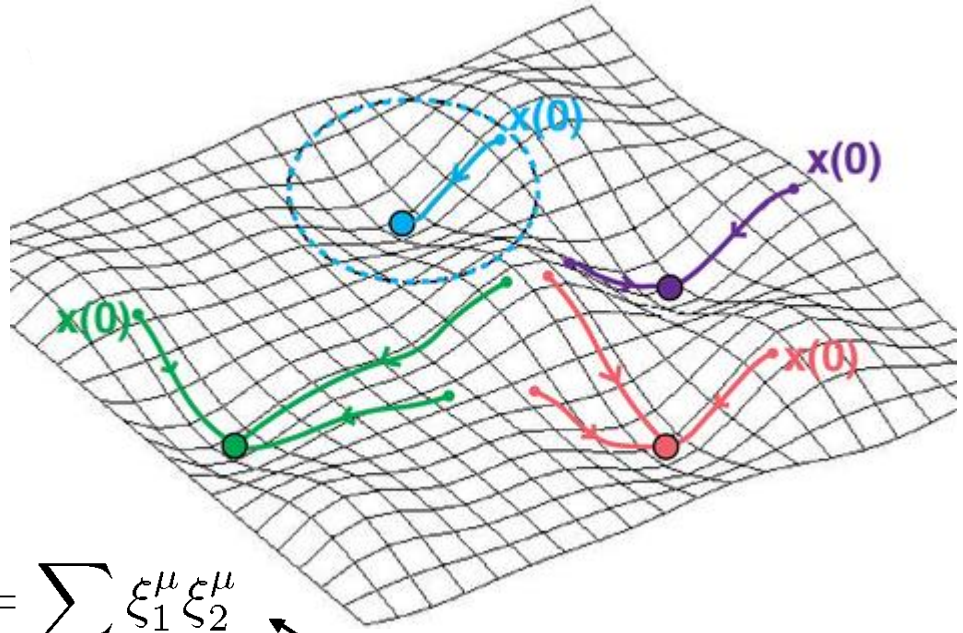
William Little proposed a similar approach in 1974.

And in 1982 the approach was proposed again by John Hopfield, who popularized the idea as the Hopfield Model.

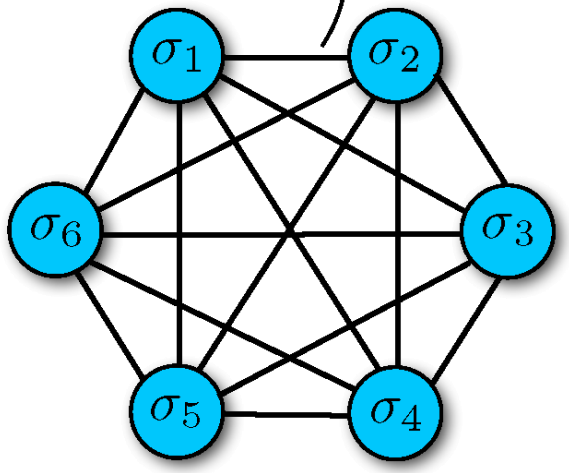
Given some set of patterns $\{\xi^1, \xi^2, \ldots \xi^N\}$of activations of our neurons, we define the strength of the connection between neurons i and j as the average outer product of each pattern with itself:

$$J_{ij} = \sum_{\mu=1}^{N} \xi_i^\mu \xi_j^\mu$$

Then set $s_i = \begin{cases} 1, \ \sum_j J_{ij} s_j \geq \theta_i \\ -1, \ \text{otherwise} \end{cases}$

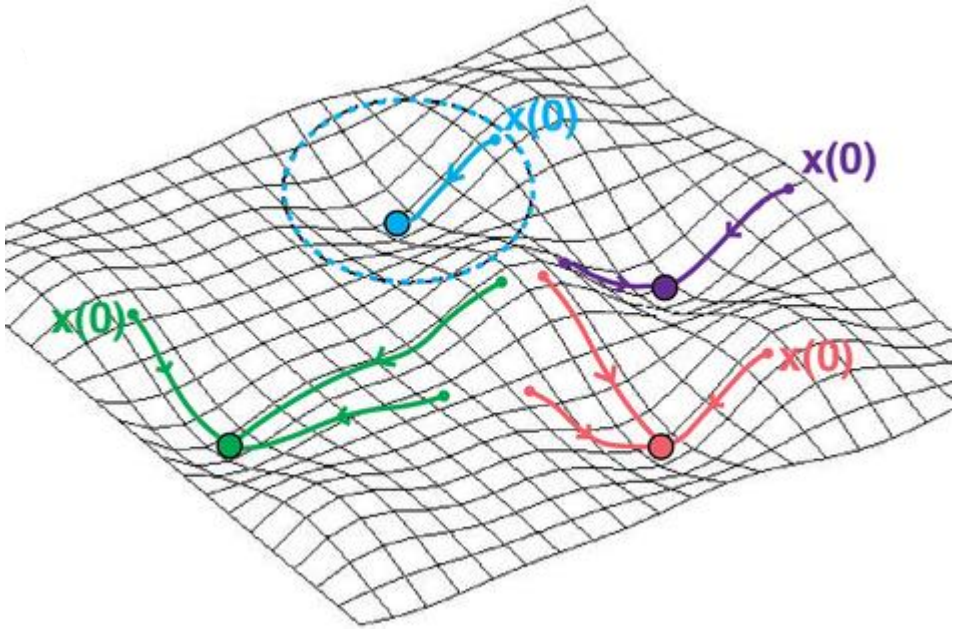


$$J_{12} = \sum_{\mu} \xi_1^\mu \xi_2^\mu$$



Ising, spin glass, and Hopfield models are all about setting a network's fixed points

We can control the steady-state solution of a network's firing rates. But do networks always have steady-state solutions? And what path do they take to get there?



Back to chaos: it's in RNNs, too

The spin glass model used the following definitions:

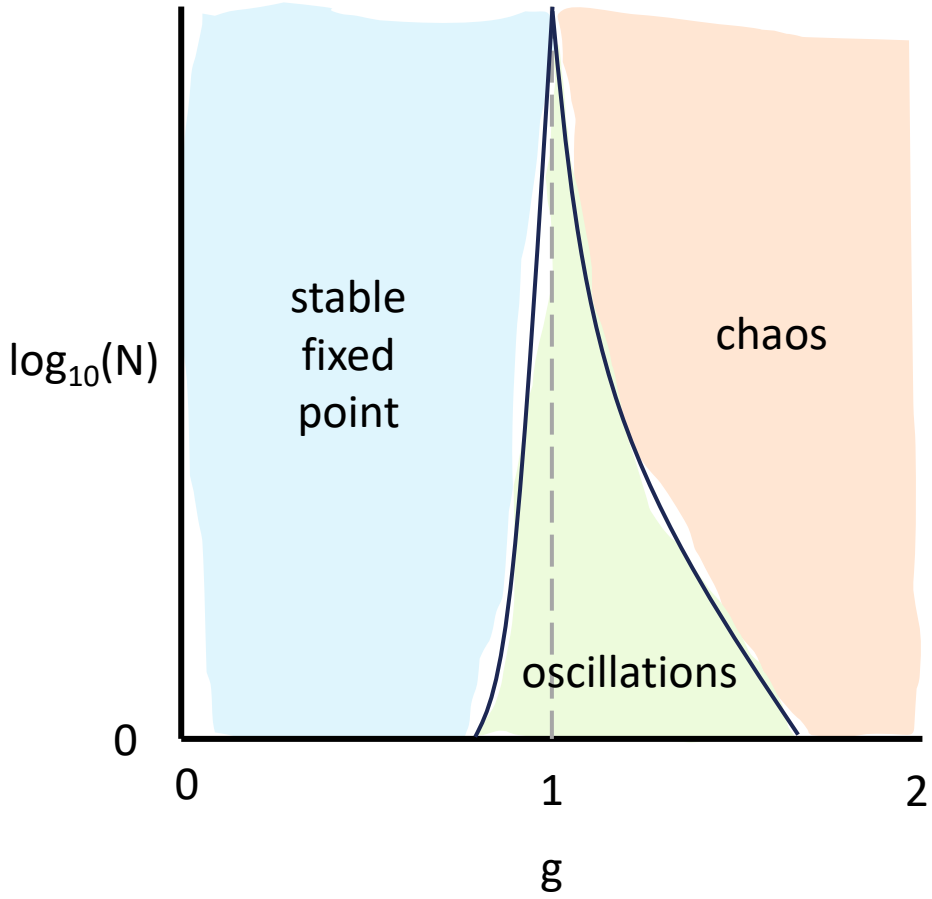
$$H(\sigma) = -\sum_{i,j} J_{ij} S_i S \qquad P(J_{ij}) \sim \mathcal{N}(J_0, J^2)$$

By the 80's, researchers were using a similar formulation to study RNNs, now also considering the dynamics of those networks. Introduced by Sompolinsky, Crisanti, and Sommers in 1988:

$$\tau \frac{dx}{dt} = -x + \sum_{j=1}^{N} J_{ij} \phi(x_j) \qquad P(J_{ij}) \sim \mathcal{N}(0, \frac{g}{\sqrt{N}})$$

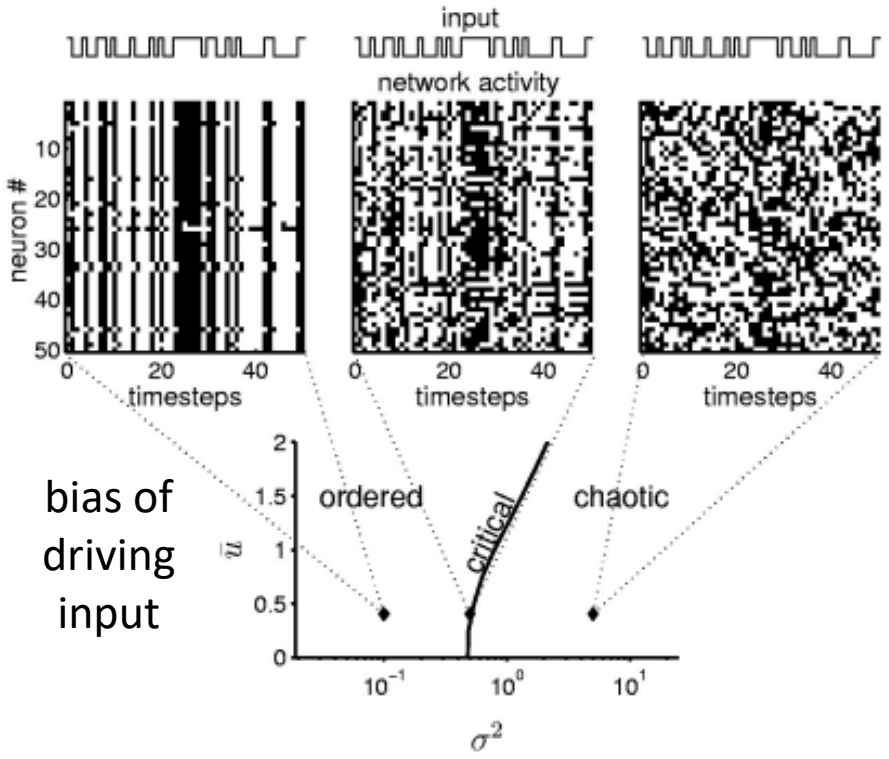
where N is the number of neurons, g is a "gain" parameter on the synaptic weights, and they defined $\phi(x_j) = \tanh(x_j)$.

In the limit where $N \to \infty$, this system has a sharp transition to chaos at $g = 1$.



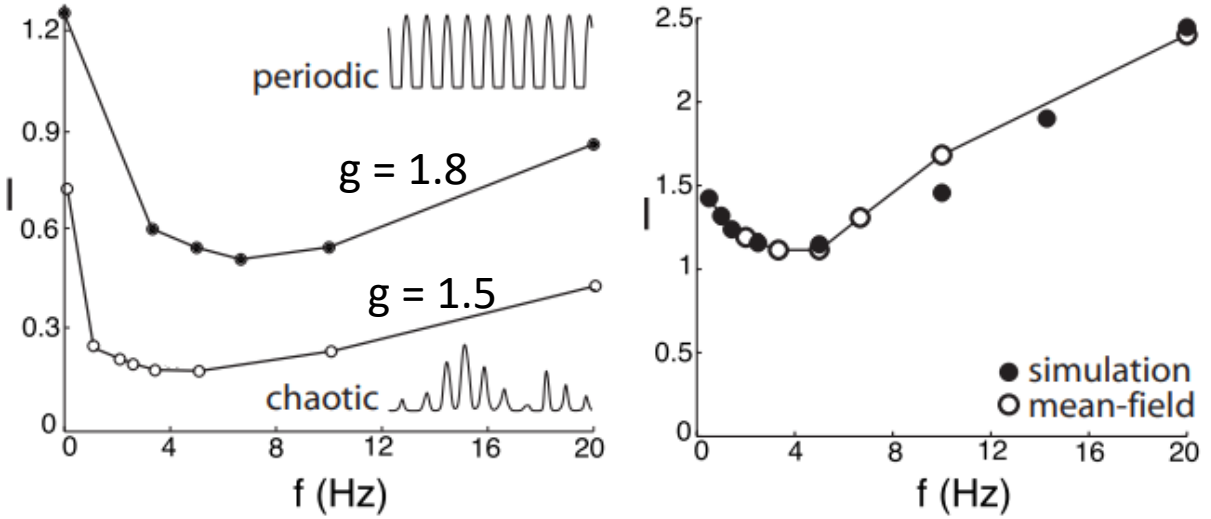
^ Ann can't find the paper that actually has this figure but it's definitely out there somewhere.

We can control the transition to chaos by driving networks with different amounts of input



variance of recurrent weights

Bertschinger and Natschläger 2004



Kanaka Rajan et al. varied the amplitude (I) and frequency (f) of a sinusoidal input to an RNN and determined when chaos was suppressed.

Rajan, Abbott, and Sompolinsky 2010

Why g=1, and what's so special about the edge of chaos?

$$\tau \frac{dx}{dt} = -x + \sum_{j=1}^{N} J_{ij}\phi(x_j) \qquad P(J_{ij}) \sim \mathcal{N}(0, \frac{g}{\sqrt{N}})$$

Recall from day 3:

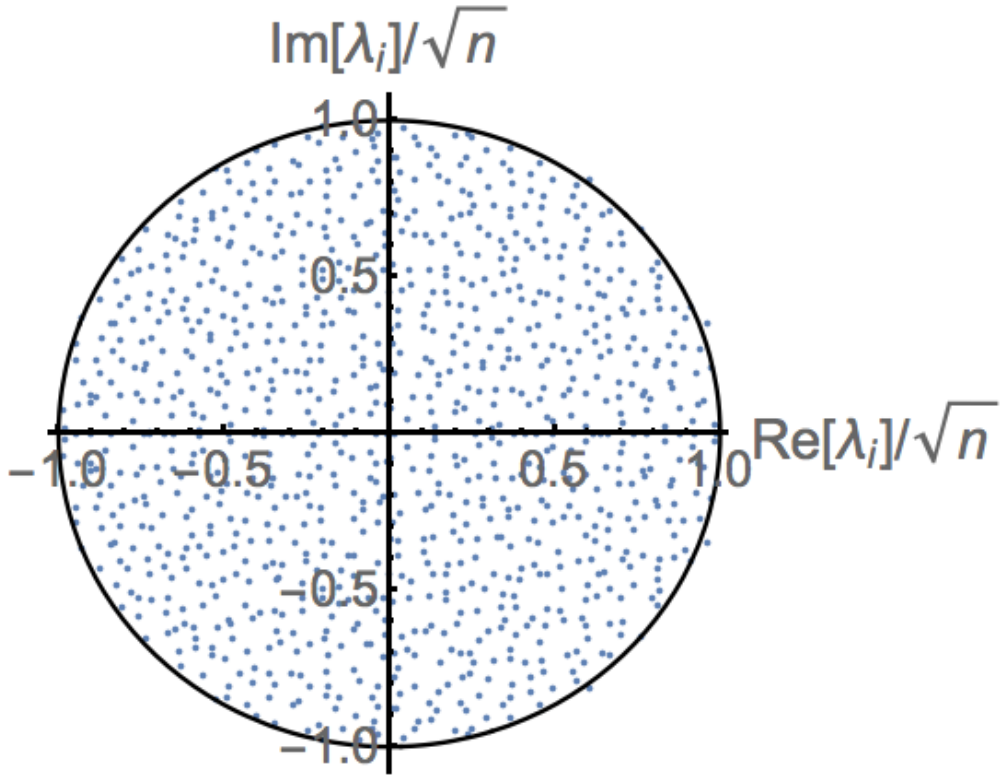
We can decompose J into its eigenvectors and eigenvalues: $J = UVU^{\dagger}$ where V is a diagonal matrix of eigenvalues.

This gives us an equivalent system of independent eigenmodes, $\frac{du_i}{dt} = \lambda_i u_i$, where $u = U^{\dagger}x$ and u_i is a column of u.

Ignoring the nonlinearity ϕ and defining u_1 as the eigenmode with the largest eigenvalue. For large N, this eigenvalue should be $\lambda_1 = g$, giving:

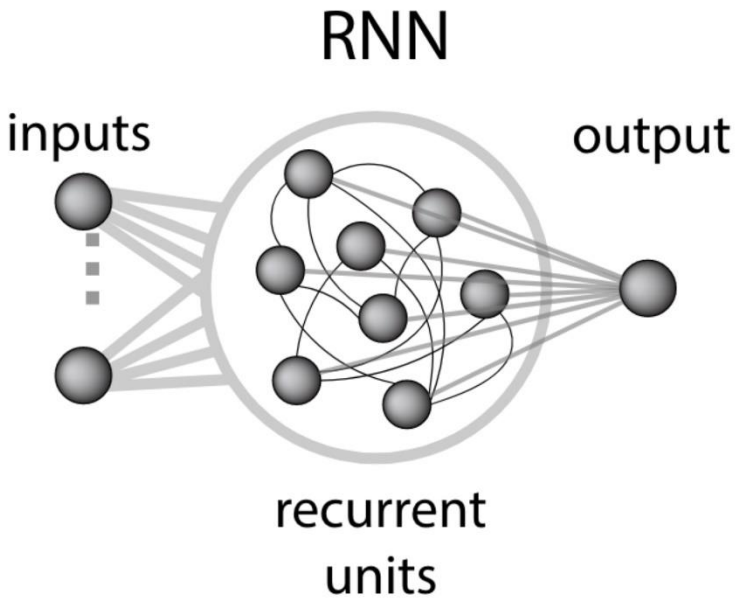
$$\tau \frac{du_1}{dt} = -u_1 + g u_1$$

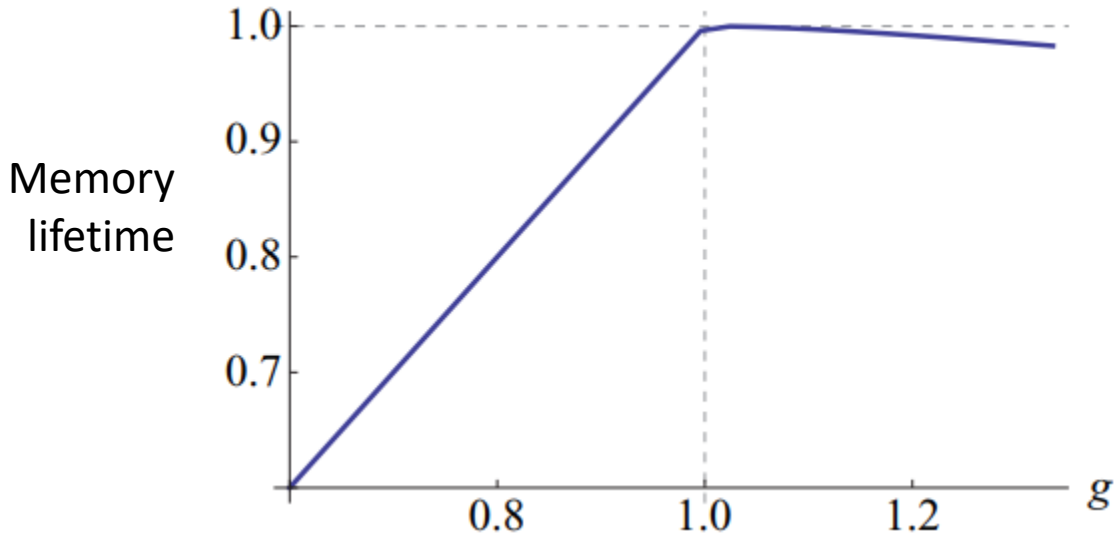
$$\frac{\tau}{g-1}\frac{du_1}{dt} = u_1$$



Random matrix theory tells us that for an NxN random matrix with independent and identically distributed entries, the eigenvalues of the matrix are uniformly distributed within a circle of radius $\frac{1}{\sqrt{N}}$ in the complex plane.

Why g=1, and what's so special about the edge of chaos?



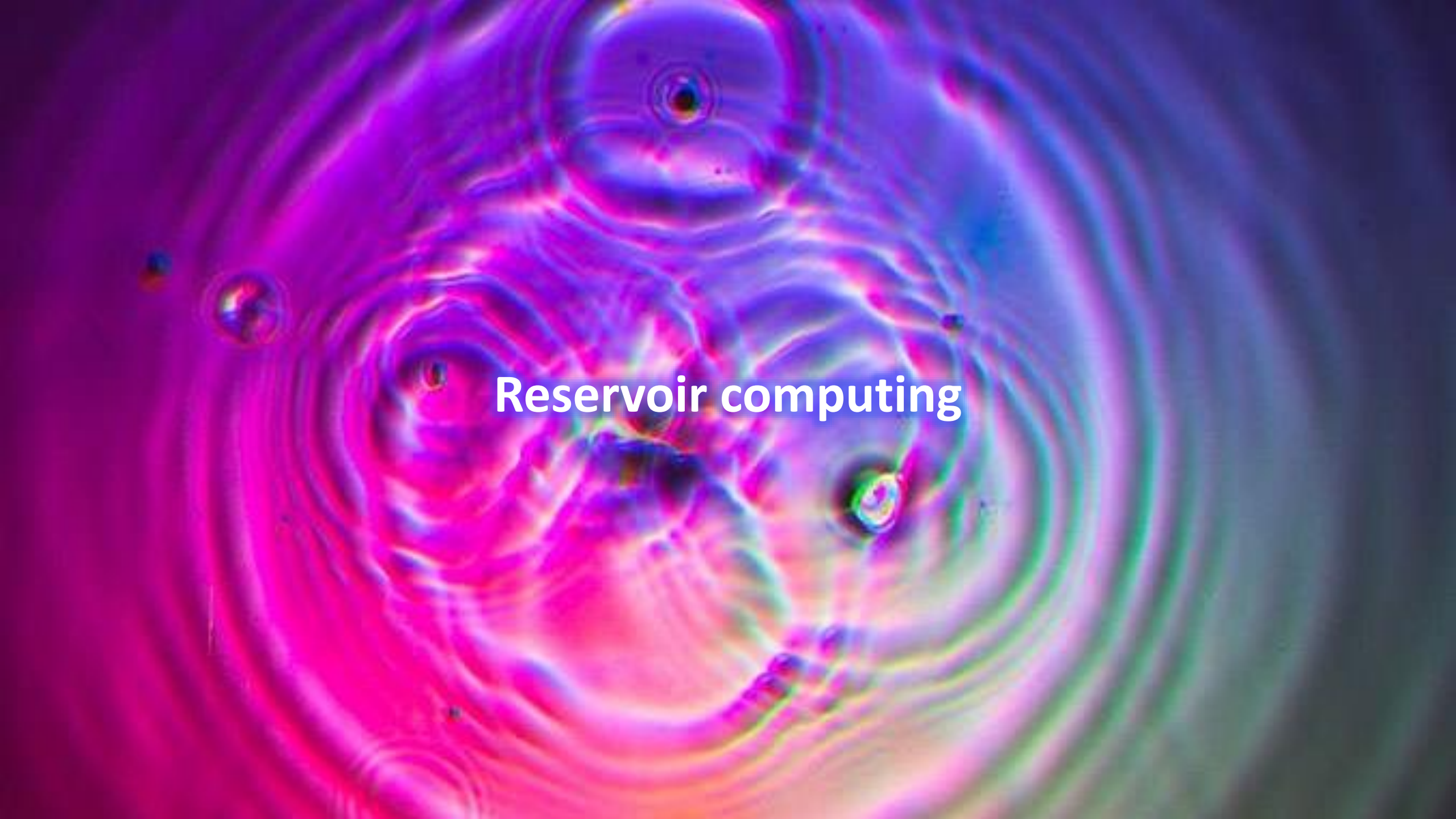


Given input at time t, how well can an optimal linear decoder reconstruct that input at time t+T?

Toyoizumi and Abbott (2011) showed that the lifetime of memory traces in RNNs peaks for g=1.

The long memory of RNNs that are poised at the edge of chaos brings us to our third theme…

White, Lee, and Sompolinsky 2004
Ganguli, Huh, and Sompolinsky 2008
Toyoizumi and Abbott 2011

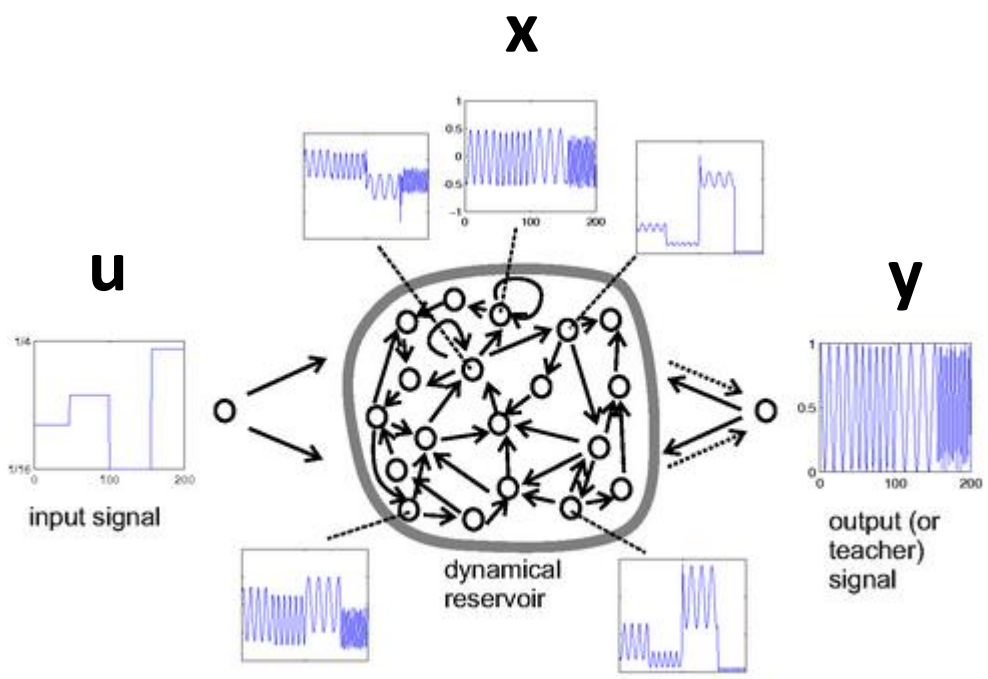

Reservoir computing

The reservoir metaphor for dynamics of RNNs

Imagine I drop a series of stones into a pool of still water. The ripples it casts are a deterministic function of the stones' properties—when they were dropped, where they were dropped, how large they were.

By looking at the surface of the pool some time later, I can reconstruct the history of its inputs from the pattern of the ripples. I could also perform operations on those inputs: for example, by looking at the radius of the ripples, I could tell you that the stone on the right was dropped before the one on the left.

Reservoir computing emerged as a new flavor of analog computing in the 2000's



$$\tau \frac{dx_i}{dt} = -x_i + \sum_{j=1}^{N} W_{ij}^{REC} \phi(x_j) + W_i^{IN} u + W_i^{FB} y$$

Echo state networks. Herbert Jaeger (2001) The "echo state" approach to analysing and training recurrent neural networks.

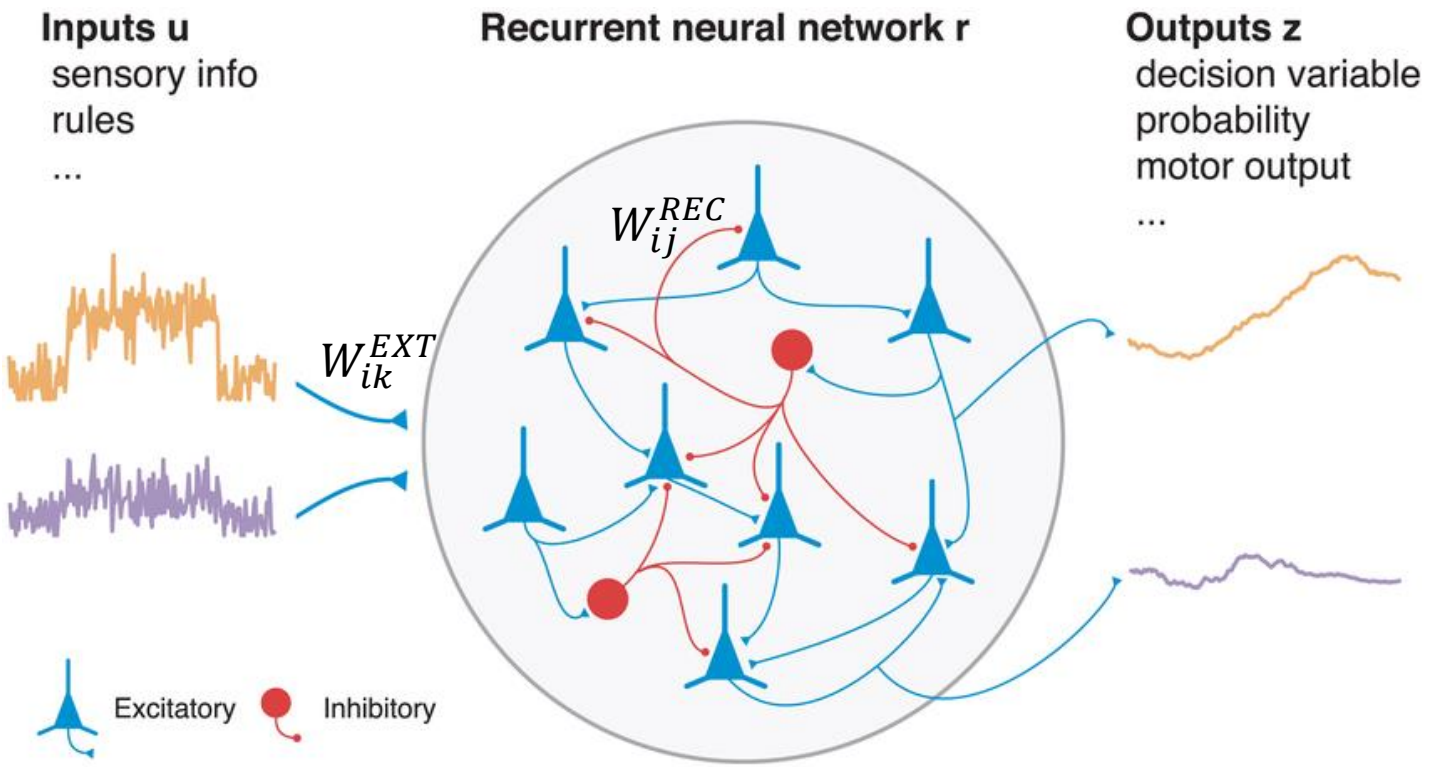
Liquid state machines. Wolfgang Maass, Thomas Natschläger, and Henry Markram (2002) Real-time computing without stable states: a new framework for neural computation based on perturbations.

Some reviews:

Buonomano and Maass (2009) State-dependent computations: spatiotemporal processing in cortical networks.

Jaeger et al (2023) Toward a formal theory for computing machines made out of whatever physics offers.

Back to where we started



$$\tau \frac{dx_i}{dt} = -x_i + \sum_{j=1}^{N} W_{ij}^{REC} r_j + \sum_{k=1}^{M} W_{ik}^{EXT} u_k$$

$$r_j = f(x_j)$$

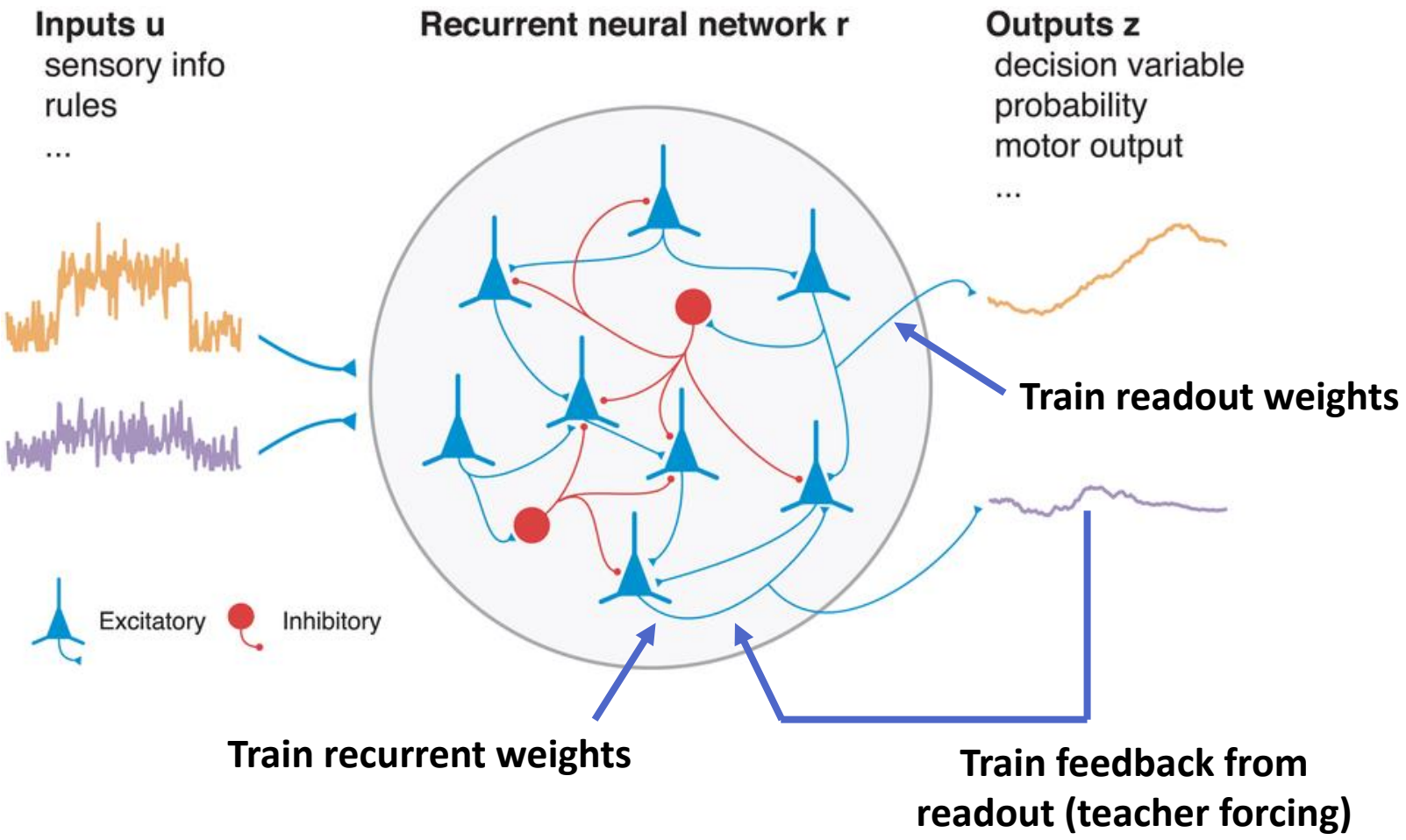
f might be tanh, sigmoid, rectified linear…

Or, in simpler notation:

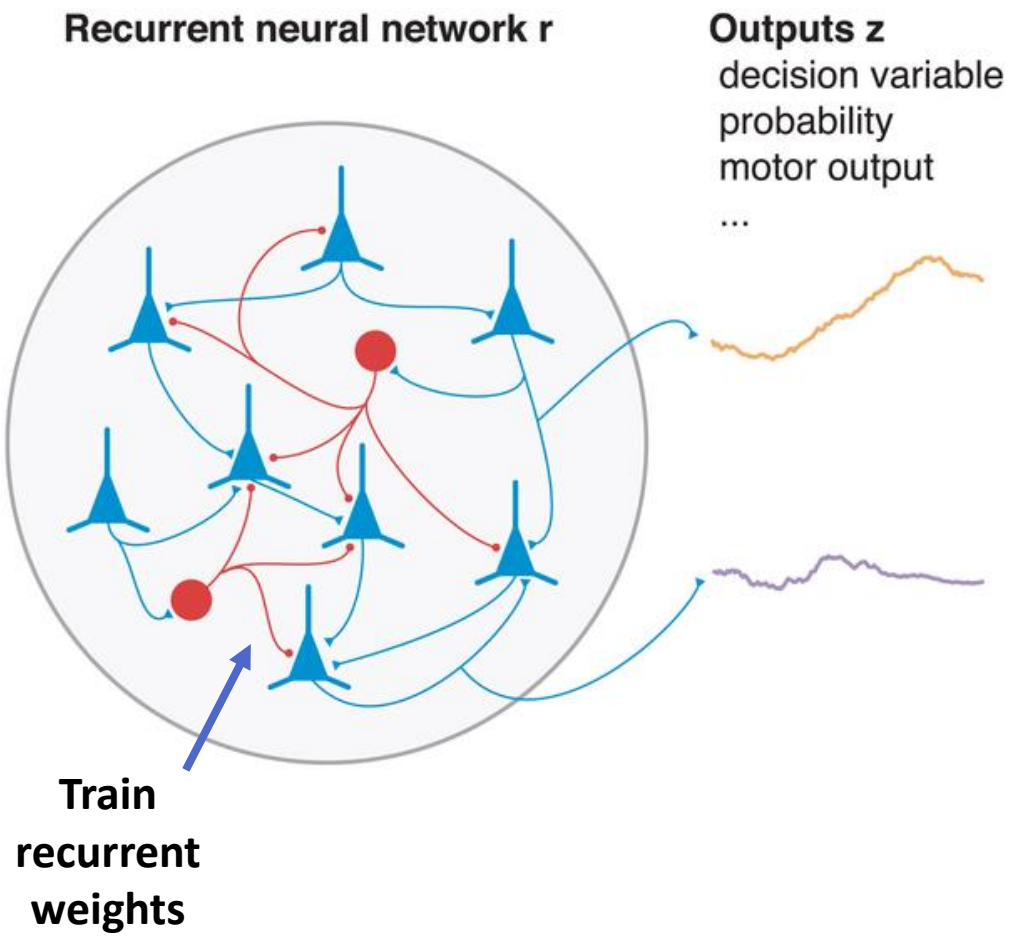
$$\tau \dot{x} = -x + W^{REC} r + W^{EXT} u$$

Song, Yang, and Wang 2016

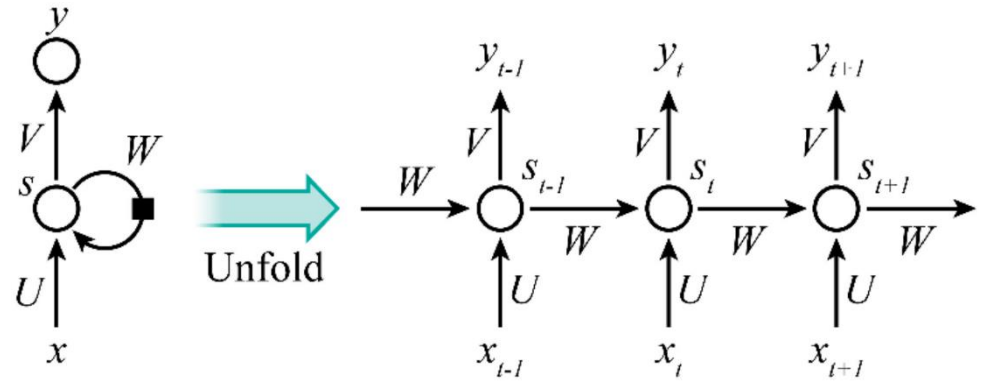
Function generation with RNNs



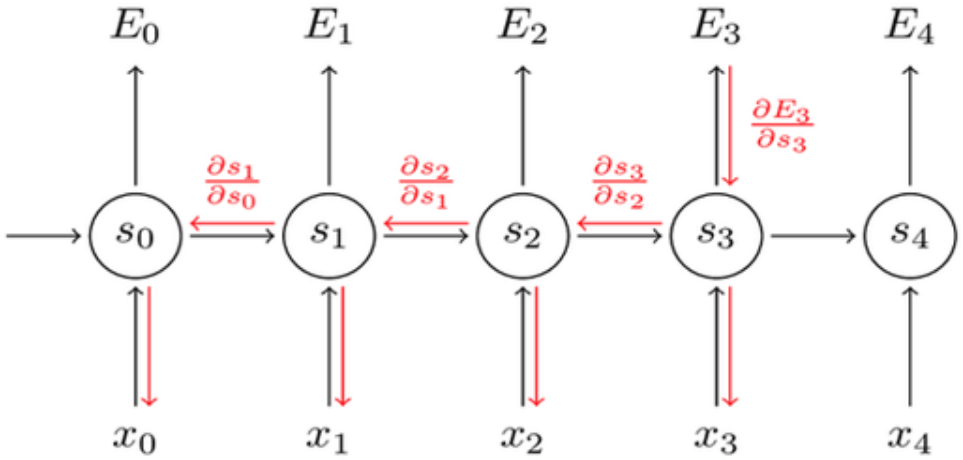
How do we know which recurrent weights to modify?



We can unroll the RNN in time to treat it as a feedforward system:

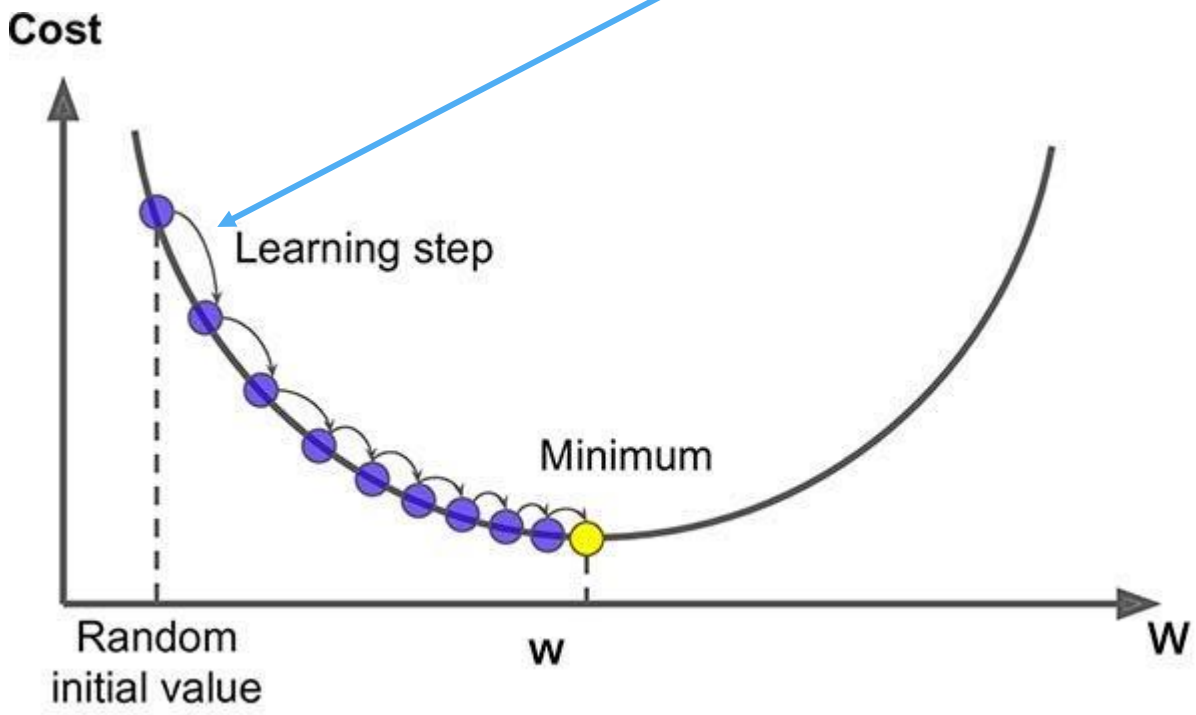


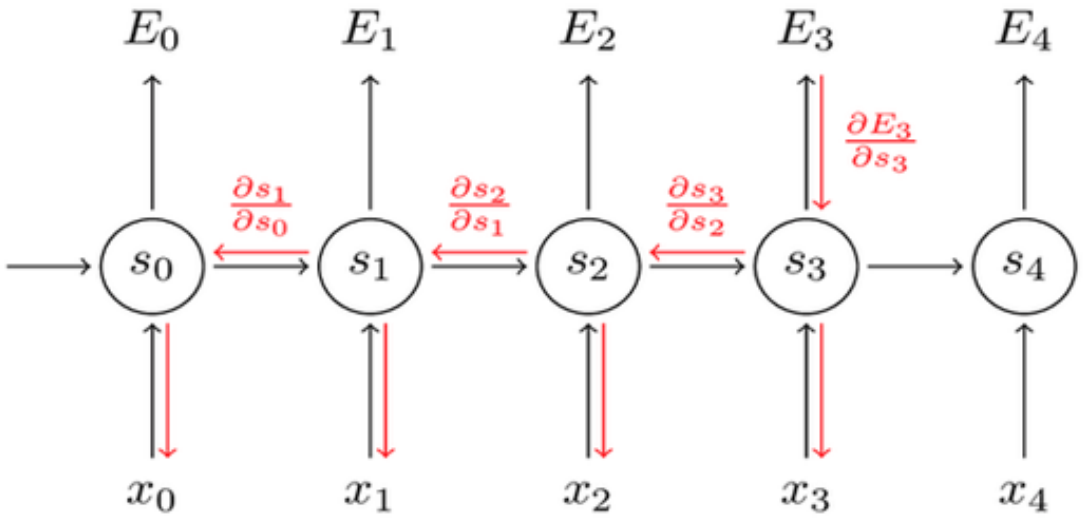
Backpropagation through time (or any method of your choice) then gives an update rule for the model weights:



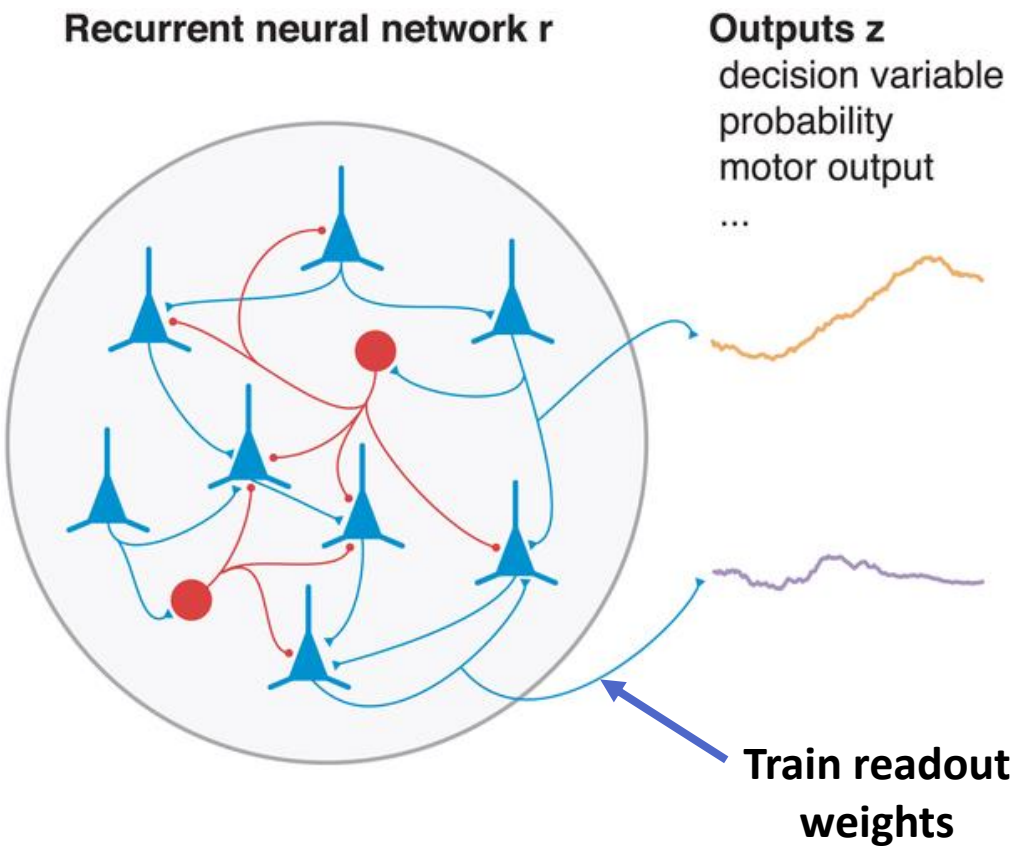
How do we know which recurrent weights to modify?

$$w = w - \alpha \frac{\partial loss}{\partial w}$$





Function generation with RNNs



$$z = Wr$$

Let's say we want to match **z** to some target function **y**. If our readout of the network doesn't feed back onto it, then we're simply doing linear regression of our neuron responses against our target, to minimize the regularized error

$$MSE = |y(t) - z(t)|^2 + \gamma \sum w_j^2$$

Recall from lecture 1 (!) that this has the solution

$$W = yr^T(r^T r + \gamma I)^{-1}$$

So our approximation of **y** is given by

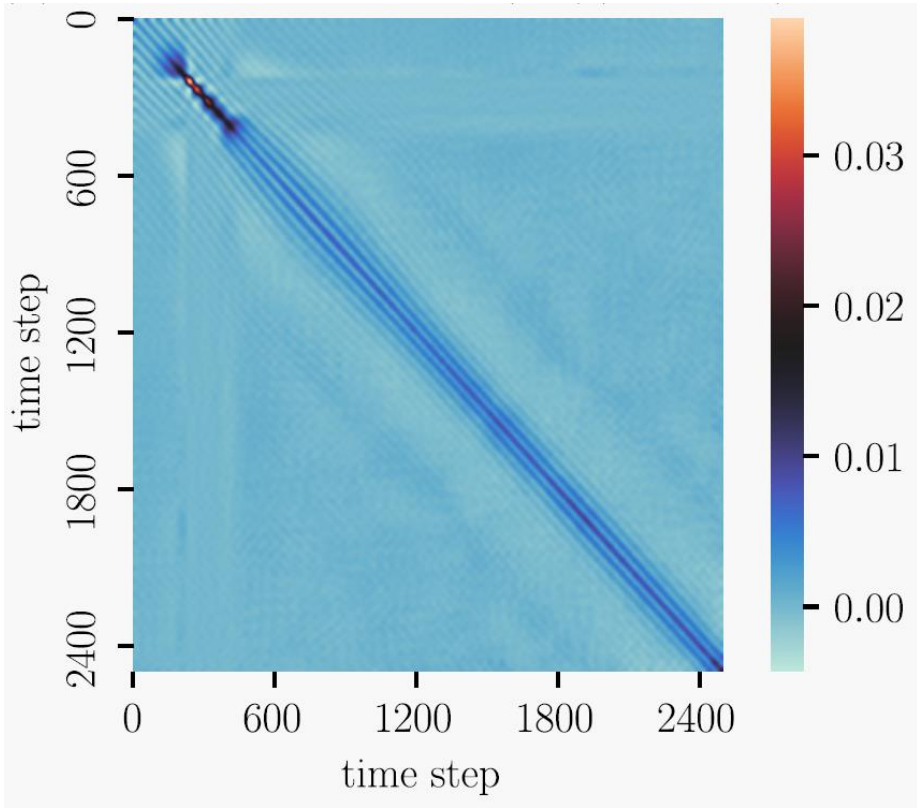
$$z = yr^T(r^T r + \gamma I)^{-1} r$$

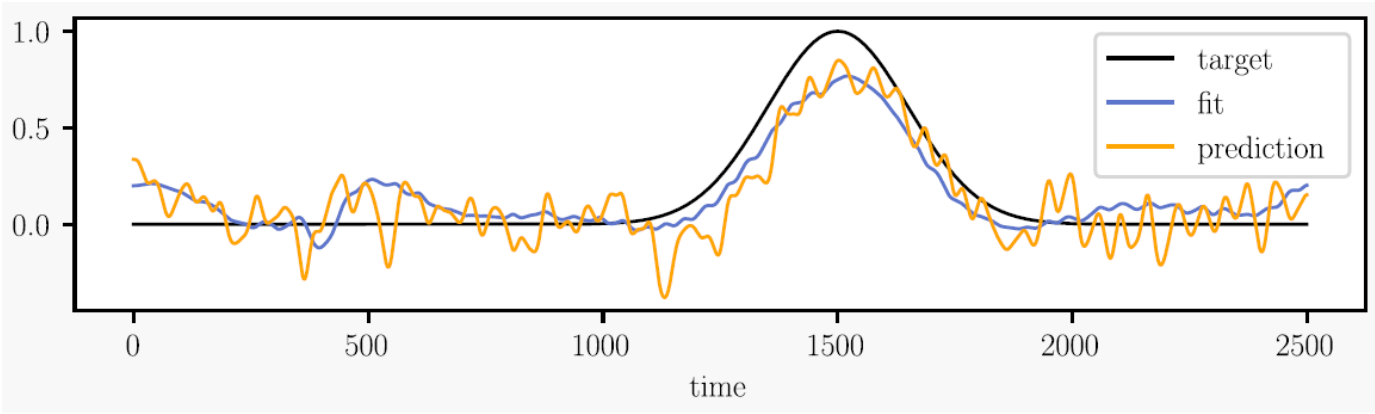
Function generation with RNNs

$$z = \underbrace{yr^T(r^T r + \gamma I)^{-1}}_{\text{N x N matrix that only depends on } \mathbf{r}} r$$

We find that our readout is simply a product of our target function **y** with an N x N matrix, which we call the network response kernel.

A network that can match any function would have a kernel that equals the identity matrix.





Function generation with RNNs