random matrices and attractors

neural networks use attractors to represent states
question: how many attractors are there in a neural network, and how do they transition between each other?
answer: orthogonal matrices maximize it. Route: networks modeled by 0, ±1 units (justified: spines) elementwise random matrix? → one attractor, by looking at eigenvalues. orthogonal matrix: all eigenvalues the same. binary matrix: different orthogonal vectors can be closer or farther from each other based on differing edit distances. Interpretations: the analysis of modal analysis of neural networks which converge to orthonormal eigenvectors are the situation with maximum number of attractors. And, biological neurons could implement a similar kind of thing

Neural networks
attractors <-> automata as computation models

- **thesis.** Computation in intelligent systems needs to be somewhat discrete for the sake of precise nonlinear computational functions and structures but also tolerant to faults of a continuous system. Asking a question about attractors yields a possible way to go about implementing this.
- **cellular automata are virtual computation**
- **NN attractors implement automata.** Neural networks use attractors to maintain states. These are important for understanding persistent computations, as in RNNs which maintain information over time iterations. Furthermore, neural networks can emulate cellular automata, which being universal computers, are a reasonable model for how a neural network can do arbitrary complex parallel computation over time. Thus, an improved understanding of these attractors of a neural network sheds light on they can change between stable states in a cellular automata sense.
- **Questions.** how many attractors does a NN have? How do they change between each other? How do the basins of attraction compare? Answering the latter two could shed light on how a NN can implement 'fuzzier' automata. **Why:** Such fuzzy automata could be powerful for resolving the trouble of discrete computation – a lack of graceful degradation, thus also being unrealistic models of biological neural computation – and continuous computation of a neural network – which lacks the precision and structure of a complex computing system.
- **Our model.** Let's model a large, recurrent neural network with several discrete values and where next activations are stepwise in inner product of weight and current activations. This models a recurrent network or Hopfield network with excitatory or inhibitory connections to each other, which is a model of the brain reasonably well. and RNNs having short term memory IS a core reason to use them! A question is, what do the attractors of this network look like?
- **eigenvalues.** What is the expected number of attractors in a network? Answer: attractors correspond to the stationary distributions of the matrix. If all p_ij of a stochastic matrix P are >0, then there is a unique stationary distribution for any and all initial distributions. However, if there are zeros allowed, there can be numerous stationary distributions. So this reduces to how many stationary distributions there are of a matrix and what they look like. As it turns out[!!], stationary distributions correspond to the maximal eigenvalue, and their numerousity corresponds to the multiplicity of that maximal eigenvalue. Thus, the number of attractors of a neural network correspond to …see Recursive Macroeconomic Theory textbook
- **matrix 1: random.** Let matrix M have elements m_ij drawn from Ber(p). That is, let matrix M be a randomly-connected finite graph. If M is large enough, then the expected eigenvalues are distributed with one large one and the rest all about equal at a much smaller amount. Thus, a random matrix M tends to have one

unique stationary distribution. Their distribution is the Marchenko–Pastur distribution, @ the incidence of new eigenvalues as p changes.

- **matrix 2: orthogonal binary.**; Let matrix M (n*n) have all its eigenvectors have the same eigenvalue. Then, M is an orthogonal ((? orthonormal? or simply normalized?)) matrix. If m_ij \in {0,1}, then that means that if m_ij=1, then m_ik=0 for all k≠j aka there can only be one 1 in any given row. There can, however, be m multiple 1s in a column in exchange for a full suite of n attractors and eigenvectors, leaving only n-m, and the remaining m are zero vectors. However, any zero vectors represent irrelevant neurons in our system in question, so we can actively ignore them and assume wlog that the binary matrix is also doubly stochastic (ie, one 1 and n-1 zeros in each row and column). This includes the identity matrix and permuted versions of it to allow cycles. In this case, the attractors(/eigenstates?) are trivial and do not interact.

- **matrix 3: nearly orthogonal binary.** Let our connectivity matrix be identity and flip some 0 to a 1 in column i (ie, connect neuron i to another j). Then the j's attractor 'subsumes' i's. (This is the case when the initial distribution is nonzero in i.) Generally, the eigenvector corresponding to the zero vector with a one at index i is now gone. This accords with the understanding that orthogonal components correspond to attractors, whereby one fewer orthogonal component corresponds to one fewer attractor and one lower multiplicity of the eigenvalue 1.

  This doesn't help our understanding substantially when we look at connected graphs: when neurons connect to each other, then they admit one attractor state, which harmonizes with the understanding of matrix P above that nonzero graphs have but one stationary limiting distribution. Specifically, while we now have a characterization of how many attractors a given graph has, and how two attractors merge. But is there a way to understand how three attractors interact when, say, neuron i grows connections to neurons j and k simultaneously?

, and how they can be merged, we don't see yet how they can be split (which is not a big deal…) nor more importantly how they relate to each other.

- **matrix 3: ternary.** xyz
- **matrix 4: Hopfield network.** xyz
- **matrix 5: real-valued network with nonlinearities.** …which in conjunction keep it from escaping/exploding. These are good for modeling biological systems! even might substitute for the time-saturation phenomenon of neural firing. Seizures.

A little empirical experiment: take a random integer-valued (1,0 valued; -1,0,1 valued…) matrix with zeros. sample many initial distributions and run them until convergence to a stationary distribution. Fit a regression model on these distributions as inputs and which stationary they correspond to as target values. Notice the size of these attractors. Secondarily, look at the perturbability of each attractor based on initial values, and compare them to the size of the attractors! [tag 14424]

Rapid prototyping and SMART writing and development

; SMARTER: Specific Measurable Actionable Relevant Timely, Ethical Rewarded/Resourced
outline of maximally concise version, which can be fleshed out later in a 2nd draft or part 2.

1. Delay any thesis or hypothesis. Simply walk through an explanation as if a podcast. Add them only in a 2nd draft.
2. Postpone discussing automata. That intuition and motivation can be added in a 2nd draft – it's cursory and auxiliary! Alternatively, making this analysis of attractors as a part 1 could render a part 2 more fluid and natural. *Knowing what we now know about a neural network's attractors, what sorts of automata could they implement?*
3. Beginning of text. Assume the reader knows what a neural network's attractors are and how they represent states. Provide a reference or two directing the reader. Postpone describing them in full for the 2nd draft; only describe them when something later specifically warrants an earlier exposition here.
4. Step 1 of the analysis: RNN network structures admit attractors
5. Step 2 of the analysis: stationary distributions of matrices is understood. Explain how the stationary distributions correspond to the eigenvalues and their multiplicities. Attractors correspond to stationary distributions of the matrix representing the connectivity graph.
6. Step 3 of the analysis: identify several kinds of matrices representing the graph of connected RNN neurons.
7. for 2nd draft / as step 4 of the analysis: talk about automata, computing with automata, state-actor invariance, distributed automata computing, how attractors are automata, and how attractors are natural for distributed automata computing
8. Further steps, for 2nd draft: do analysis of matrices (empirical or analytical) of the attractors' shapes, radii, and perturbability/stability. [tag 14424]

;

;
Title: Understanding neural network attractors

It's well known that recurrent neural networks can maintain states via attractors. These attractors have the potential to perform complex computations (see fuzzy automata discussion later). This exposé asks: How many attractors are there in a neural network? By examining these attractors, we improve our analytical understanding of the RNN's building blocks of complex computation and human cognition.

To outline how answer this question,

1. First we explain what we mean by neural network attractors and why they matter.
2. Second, we discuss stationary distributions of matrices, their properties, and how they relate to neural attractors.
3. Third, by analyzing specific matrix models of recurrent networks with theories of stationary distributions and eigenvalues, we can illuminate some qualities of these attractors.
4. Fourth, we relate attractor dynamics to automata, distributed processing, and complex computing.

Attractors naturally arise in recurrent neural networks under mild conditions. They serve as models for learned associative memory and short-term memory states, and they've found widespread use particularly in work on Boltzmann machine and Hopfield-style neural networks. Attractor-based persistent memory states endow the network with the building blocks for complex computation and high-level cognition in the [cortex]. We've known how useful persistent memory states in neural networks are for a while: LSTMs, for example, explicitly build in memory systems via states that persist over time, using neurally-controlled gating methods, which helps them utilize information over long timescales and distances. And non-distributed models of cognition or any kind of complex processing usually have some kind of state-storage system. [act-r?] In this exposé, we'll examine the attractors that automatically arise in simple RNNs.

In the basic setting, neural networks consist of layers with input and output vectors, and the neural computation can be represented as a matrix operation (plus nonlinearity) between these inputs and outputs. Recurrent neural networks additionally connect neurons recurrently back to themselves, so that a hidden output to a layer becomes a hidden input to the same layer on the next timestep. We can represent a recurrent layer's processing with the canonical formula: $[y, p_{t+1}] = \sigma(M[x, p_t])$. If we merge the inputs to each other and the outputs to each other, ~~If we combine the external afferent input and the hidden input into one vector and the efferent output to the hidden output,~~ we can simply say without a loss of generality: $p_{t+1}=\sigma(Mp_t)$. Alternatively, we could rewrite this $p_{t+1}{}^{T}= \sigma(p_0{}^{T}M^{T})$. ~~Since M isn't specified yet, we may as well say $p_{t+1}{}^{T}= \sigma(p_0{}^{T}M)$.~~ Let's now think about two timesteps of this recurrent neural system: $p_{t+2}{}^{T}= \sigma(\sigma(p_t{}^{T}M^{T}),M^{T})$. If we ignore the nonlinearity, we can see that in general, $p_{t+N}{}^{T} = p_0{}^{T}M^{T}M^{T} = p_0{}^{T}(M^{T})^2$. This even makes sense in a limit: $p_{inf}{}^{T}= p_t{}^{T}(M^{T})^{inf}$. This limit exists under certain conditions on p and M, which we'll get to later.

[An attractor p is a state that is unchanged by the action of M, and other state vectors close to p will also settle to that same attractor center p; the set of vector-states that eventually settle to p is known as the basin of the attractor.] That is, attractor states satisfy $p_{t+1}{}^{T}= p_t{}^{T}M^{T}$…? Attractors thus stabilize states with noise or errors that are nearby p. In the cognition interpretation, if a person is holding a noisy or incomplete 'thought' as a state vector, the neuron-based neural network will automatically tweak the thought to completion.

But attractors can do more than correct errors: they can hold informational states. States are important for complex computation. Every digital computer has a core is based around state registers and the actions that can be done with those states. And humans think using interacting logical components. So by examining the recurrent neural network's attractors that potentiate these stable states, we can gain a better understanding of how neural networks can give rise to building blocks of complex processing.

[[[For example, one attractor could be a 'dog' attractor. That is, the state vector associated with the attractor holds all the information about the dog: it has two ears, its legs have these joints, etc. /// Let's say someone sees a dog. They'll probably only see some of its legs and only one eye from their angle, but these would

enough components of the state vector being related to dogs. The network would automatically settle it to the basin of dog, which holds the information about all the legs. (This is known as 'pattern completion'.) Another example would be to think of two basins as bits: depending on which basin the state vector lies in, the layer holds the value 0 or 1. In circuitry, flip-flops operate on a similar principle of holding data over time via recurrent connections – clearly, collections of flip-flops are powerful enough to display this digital text you're looking at or drive the printer that it's printed on!]]]

Vanilla neural networks don't have attractors – they process information in layerwise feedforward connections and each of those layers are different. Fortunately, any model of biologically plausible cognition uses recurrent neural networks or other networks that feed information back into itself.

The RNN timestep's form $p_{inf}^T = p_t^T (M^T)^{inf}$ resembles something from the theory of dynamical systems and Markov chains. Let's say that vector p and the rows of matrix M are stochastic: all the entries are strictly positive and sum to 1. If $p^T = p^T M$ ($= p^T M^2 = p^T M^N$!) then we call p a stationary distribution under the transition matrix M. This vector state p is unchanging, or stationary, when it undergoes the transition between states defined by M. [Give a simple numerical example.] By the Gershgorin circle theorem, there always exists a ~~unique~~ stochastic stationary distribution vector for these stochastic p and M. ~~If the entries are picked randomly, then this stationary distribution is unique.~~ Furthermore, for any other initial stochastic distribution p_0, repeatedly applying M will always eventually converge to that unique stationary distribution. (This is a form of ergodicity.) From linear algebra, you might notice that the p which satisfies $p^T = p^T M^T$ is precisely the eigenvector of M whose eigenvalue λ is 1, given by the usual eigenvector/eigenvalue relationship, $Mp = \lambda p$. (That the unique stationary distribution is an eigenvector is shown by applying the Brouwer fixed-point theorem.) ~~What this means is that a normal RNN layer with mild conditions always has a stationary distribution defined by an eigenvector.~~

Sometimes, there can be more than one stationary distribution. This can happen we alter our stochastic matrix M to allow zero-valued entries, or ~~equivalently,~~ when M more than one eigenvector that has an eigenvalue of 1. We'll come back to what happens when there's more than one stationary distribution and matrices that can have more than one. One comment now is that an NxN matrix has at most N eigenvectors, so it can have up to N attractor states, and the more eigenvectors are orthonormal, the more attractors it can maintain. But for now let's revisit neural network attractors and how they relate to stationary distributions.

We can directly our apply stationary distribution knowledge to RNN attractors. If a recurrent neural network repeatedly activates its own neurons according to M, then it has the form of $p(M \circ \sigma)^N$ where \circ denotes function composition. When M is conditioned as a stochastic matrix, or when M takes on any real values and σ is a function with image between 0 and 1 (such as softmax or heaviside step) approximately (ie, the quasi-linear matrix has eigenvalues no greater than 1) (or the Lyapunov exponents are negative) (todo, verify), then M satisfies conditions for convergence for any input distribution p, and we can transfer concepts from stochastic matrices to here. This implies that our recurrent layer has a fixed point and each input eventually converges towards one of them under repeated iteration. (Interestingly, this ought to hold for any partially trained network, even before it is trained to saturation to an optimal orthonormal eigenbasis. Intuitively, this corresponds to how a network or child learns modes in sequential order of most significant principal component; a network that has learned the first x% several modes corresponds to having x% several eigenvalues at about 1!)

So, what can we conclude about RNN attractors? ~~We know from other work that recurrent networks inherently admit attractor structures. Our new understanding of how recurrent networks model stationary distributions shines light on how these happen.~~ A normal RNN layer with mild conditions always has a stationary distribution defined by an eigenvector. Each corresponds to an attractor basin's center, and states within them are capable of maintaining persistent informational states even under small deviations(technical word?). Since our theory of stationary distributions tells us that the maximum number of stationary distributions directly corresponds to its eigenvalues, we can assert that for an NxN matrix, which has at most N eigenvectors

with eigenvalue 1, that a recurrent network with N nodes can have at most N distinct attractor states. And the more eigenvectors are orthonormal (which corresponds to more eigenvectors having eigenvalue 1), the more attractors the network can contain. Cool! This corroborates work in Hopfield networks that say an N-large network has memory capacity O(N) ... albeit not 0.15N (hopfield, 1982) ($1/2\pi = 0.159$? No, n/[2log_2(n)]). Also, from [saxe?] we know that real-valued neural networks tend to learn a progressively orthonormal basis of connectivities in steps, which would imply that a network progressively attains more attractors as it learns, by learning one attractor fully before starting the next one, which ought to be independent and not affect any of the previously learned ones.

If there are zeros in our network's connectivity matrix, which in the context of connectivity matrices means there are neurons who are virtually not connected or have no synapses between them, then we can get more than one eigenstate / attractor state. A simple example would be to make a block matrix of connectivities, where two clumps of neurons are fully stochastically connected within the clump but are fully disconnected between the clumps. These would then have two distinct areas for states to persist. Generally speaking, with zeros, there are many possible stationary vectors. A 2x2 identity matrix of connectivities will be stationary in any input vector. And if the matrix elements have real values between 0 and 1 but the matrix is overall not stochastic, we can get weaker senses of 'clumps', where, say, two different input vectors might feed into one or the other clump into a nontrivial stationary state while the other clump decays its activations to zero. Another matrix to consider could be one with oscillating states: a 2x2 identity matrix rotated by 90 degrees (with 0s on the two diagonals and 1s on the two off-diagonals) would take any input vector and sned it to a 'stationary' set of states which are not individually stationary but oscillate between two states. But for the next bit, let's assume that our matrix yields a finite number of stationary states, has no oscillations, and each input vector converges to precisely one of these states deterministically. As we'll see, these alternative examples won't cause problems.

But when does an N-neuron RNN layer actually achieve all of the potential N attractor states? For this, let's look at some specific neural systems and analyze what sorts of attractors they'll have.

Some example neural systems and their attractors

Our first neural system is a set of N neurons with connectivities simply either 0 or 1 – a binary matrix. Let each element m_{ij} be 0 or 1 according to the Bernoulli distribution Ber(p) for any p and some nonlinearity function that thresholds output values above 1 to 1 and below 0 to 0 and then normalizes the entire output vector, such as a softmax function <y_i = e^x_i / sum_j e^x_j>, cross-entropy, or a one-hot 'approximation'. According to XYZXYZ, this connectivity matrix is expected to have precisely one eigenvalue with value 1 and have the remaining eigenvalues be approximately equal in a value much less than 1. This means a collection of neurons randomly connected via Ber(p) will usually have one predominant attractor (which is centered at the individual neurons with the most inputting connections, regardless of any clustering) and any other [componentwise?] attractors will likely be much smaller in size (ie, in its correlation to input's values and dimensions). This works intuitively, too: the neurons with the most input connections are most likely to be activated are the ones who receive the most possible activations, and all else being uniform, they will be the last states to persist since they will eventually edge out any other neurons as the distribution biases more and more towards that state/neuron/attractor.

Our second neural system is another binary matrix, but this time, the rows are orthonormal. By our understanding of eigenvectors, every eigenvector has eigenvalue 1. However, for two binary vectors to be orthogonal, only one of them can have a value of 1 in any given coordinate. So while they do admit many attractor basins in theory, these attractors are independent and non-interacting. In this case, every distribution is stationary, since non-interacting clusters don't change the values per forward propagation.

Our third neural system is a real-valued matrix whose rows are stochastic and whose columns only have one nonzero entry and can be reshaped into a block matrix. We enforce that the output vector is normalized via, eg, softmax. Its stochasticity keeps the eigenvalues smaller than 1 and the states as 'attracting' instead of repelling. Each block forms a clump that's disconnected from the others. However, within each of these clumps we have an

('internally') stochastic system that looks like any other proper stochastic system with a single stationary distribution. What's different this time is that the total output vector is normalized. This means that the sum of input components for the first clump end up weighting the final stationary distribution it converges to according to the weights collectively in each other clump. However, while the stationary distribution of each clump has a characteristic distribution of its relative values, these values are each scaled according to their contribution to the total values in the whole network layer.

Our fourth neural system is connected binary but

Our next neral system is a 'ternary' (hopfield?) network

Our next neural system is a general Hopfield/Boltzmann network. These are networks whose weights are zero on the diagonal and are real-valued, with a standard perceptron activation nonlinearity function. It was in these networks that the idea of attractors first arose, I think.

==========

If a given RNN's NxN connectivity matrix has K≤N stationary distributions which biject to K attractors and attractor basins,

the phenomenon that a set of neurons which connect to each other act as a (time-homogeneous) Markov chain with respect to their connectivities. First let's look at the activity of recurrent neural networks over timesteps. Given an input activation $[p^0, p^1, p^2, \ldots]^T = p_0$, the network's connectivity matrix M acts on this input to create a new output y, up to an elementwise nonlinearity on each value y_i of y. This corresponds to a collection of neurons receiving input activations in their dendrites, accumulating these values at the axon hillock, and nonlinearly deciding to fire or not depending on that accumulated value. For a single neuron with weights m_1, we'd write this as $\sigma(m_1^T p_0) = \sigma(m_1^0 * p_0^0 + m_1^1 * p_0^1 + m_1^2 * p_0^2 + ...)$. Here, σ is the nonlinear activation function such as a connectionist sigmoid function, a deep learning RELU function, or a perceptron heaviside step function, and it corresponds to the procedure at the axon hillock. We can talk about a collection of neurons with their own individual set of weights in terms of $\sigma(Mp_0)$ or, equivalently, $y=\sigma(p_0^T M)$.
// The result y of $\sigma(p_0^T M)$ is the result of the neurons processing the input once. In a recurrent network, the weights feed back into themselves, and the output generated at the previous moment or timestep becomes the input at the next timestep. So, let's now think about the next timestep of this recurrent neural system, in which it performs $\sigma(y^T M)$. This is the same as $\sigma(\sigma(p_0^T M)M)$. If we ignore the activation (which would be totally fine if our weights were all nonzero and the nonlinear activation function were RELU!) then the 2nd timestep result would be simply $p_0^T MM$ or $p_0^T M^2$. We can extend this idea to N-many timesteps by writing $p_0^T M^N$. We can even think about an infinite number of timesteps as $p_0^T M^{\infty}$.

Now let's think about attractors. Generally speaking, attractors are basins where different inputs which differ slightly all descend the same (convex) basin given time, like a golf ball in a mixing bowl. They arise often in dynamical systems and control theory. In neural networks, the interpretation is that slightly different input activations can still converge to the same . They fully explain how recurrent neural networks can exhibit pattern completion, or the ability to yield a full pattern based on only some of the components.

The definition of a stationary distribution (or vector) p of a matrix M is the distribution that satisfies $p^T M = p$. This implies also that $p^T M^N = p$ and $p^T M^{\infty} = p$. For this to make sense, our M needs to converge under infinite composition. For certain classes of matrices, this is guaranteed to happen. From the theory of stationary distributions, we know that if M is a finite matrix, then there is a finite number of distinct p, up to normalization, that satisfy this relationship. We also know that all initial distributions p_0 eventually converge to one of these p in the limit of $N \rightarrow \infty$.

As usual in defining neural networks, let p_0 be an input vector and M be the transpose of activation matrix of a set of neurons. One forward propagation is $\sigma(M^T p_0)$ or $\sigma(p_0^T M)$ without loss of generality. In an RNN recurrent neural network, these outputs are fed back into the network in the next timestep. At time t=2, we get $\sigma(p_1^T M) = \sigma(\sigma(p_0^T M)M)$. If we ignore the nonlinearity for a moment, this second recurrent propagation takes the form $p_0^T M^2$. As t grows, we can continue this procedure: $p_0^T M^t$. As t goes to infinity in the limit, the quantity $p_0^T M^t$ converges for any p_0 to some $p_t^T M^{inf}=p_t^T$ (under suitable conditions such as M's entries taking values strictly between 0 and 1). Such a p_t is called a stable distribution. When M is a stochastic matrix, p_t is unique. In dynamical systems, p_t is also known as an *attractor*.

- RNN attractors, high-level

Attractors can be thought of as basins of convergence. An attractor could be a mixing bowl. A marble dropped on top will converge at the bottom of the bowl no matter where above the bowl it was dropped from. A system might have multiple attractors, such as a marble dropped on an egg carton. Generally speaking, time-varying states (such as position of the marble) that are close to a center of attraction will converge to that center over time when repeatedly subjected to a time-invariant transition matrix procedure (such as gravity).

Recurrent neural networks can exhibit attractor behavior. These neural attractors work by connecting neurons to each other in a way that they repeatedly activate each other. Here, the time-varying states are the inputs/output vectors of activations and the transition matrix is the connectivity strength matrix between neurons.

(Note: This notion of attractor basin descent is different than the hill-descent phenomenon of gradient-based backpropagation learning in neural networks!)

which results in activation patterns persist through time. Thus, attractors enable networks to maintain informational *states*, and they have been proposed as mechanisms for human short-term memory. Furthermore, these attractors can be connected so that even if an input is noisy, if it is close enough to the noise-free stable attractor, it will converge to that basin anyways. In this sense, they are a mechanism for cognitive pattern completion, a phenomenon observed both behaviorally and in neural models.

Let's model our recurrent neural network as having a stochastic matrix of connectivities and initial inputs being distributions that sum to one. Under these conditions, then repeated recurrent forward propagation applied to any initial input distribution will eventually converge to an attractor.

- RNN matrix form limits
  see above, after rework
- = stable distributions
  see above, after rework
- stable distribution attractors in stochastic matrices
- but RNNs aren't stochastic matrices with distributional inputs
- stable distributions theory & eigenvalues
- simple examples: attractors of identity matrix, of 2D rotation matrix, of manual one-hot matrix
- attractors of 0-1 matrices overall (+ null spaces? no nevermind)
- attractors of Bernoulli random 0-1 matrices.
- attractors of Hopfield network

'orbital' geometrical distribution of attractors in the simplex of input distributions, determined by proportional scaling or by truly uniform density or normalization of simple Cartesian uniform density… see storkey1999basins.pdf

See also PDP chapter 6 on harmony theory, standard work on Boltzmann machines, Lyapunov functions & exponents, ergodic theory,
Camden requests it once its done