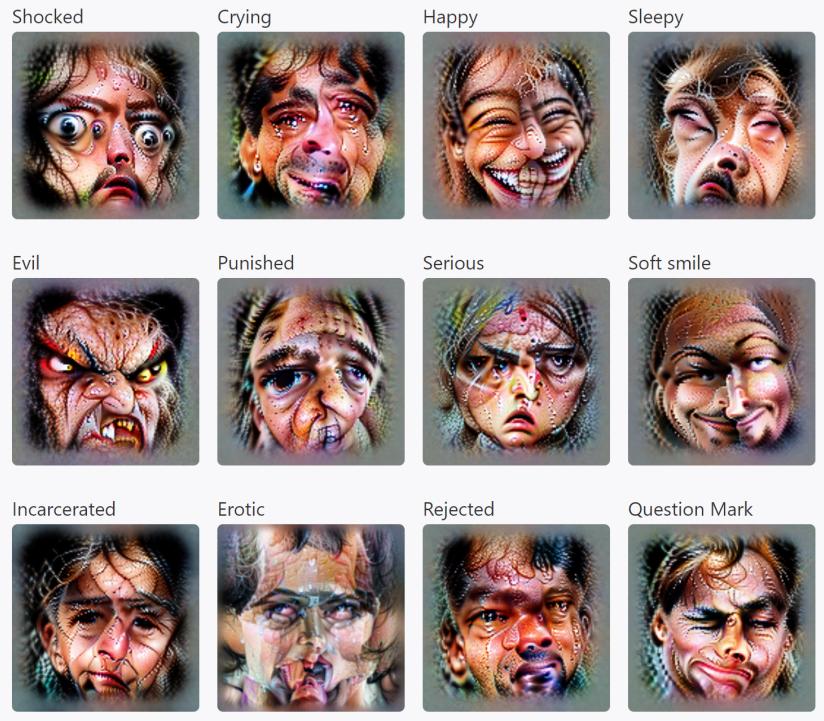
RNN Initial Hypotheses

Amrut, Ankit

April 2021



OpenAI discovered the essence of emotion (https://distill.pub/2021/multimodal-neurons/#emotion-neurons).

1 Overview

What we did:

1. Trained Recurrent Neural Networks for two different tasks using Gradient Descent (10 networks per task). We call these tasks the "perception task" and the "decision task".
2. Confirmed that the networks properly performed their assigned tasks.
3. Analyzed the changes in the network weight matrices and dynamics due to the training process.
4. Identified a few key hypotheses about the training that are worth exploring.
    - The training process increases the non-linearity of the dynamics.
    - The trained network weights are not very different from the randomly initialized one, but the few differences make large changes in relevant measures.
    - The dynamics associated with the computation being done is moved away from the readout direction to increase stability in the output.
    - The maximally predictive latent spaces are 'mostly unique'. i.e. Optimizing to find the maximally predictive subspace is a 'mostly convex' optimization problem.

2 Network Training

The neural network used for the perception task contains 128 nodes, while that of the decision task contains 256 nodes. Both networks are fully connected.

Network activity approximates the continuous time equation,

$$\tau \frac{d\mathbf{r}}{dt} = -\mathbf{r} + f(\mathbf{W}_{rec}\mathbf{r} + \mathbf{W}_{in}\mathbf{u} + \mathbf{b} + \sqrt{2\tau\sigma^2}\mathbf{N}(0,1))$$

In our simulations, $\tau = 100$ms was the neuronal time constant. In addition, $\mathbf{u}$ is the input of the network, $\mathbf{b}$ is the bias of each node, $\sigma = .01$ is the strength of the noise applied to each node, and $\mathbf{N}(\mathbf{0},\mathbf{1})$ is a Gaussian white noise process with mean 0 and standard deviation 1. $\mathbf{W}_{in}$, $\mathbf{W}_{out}$ and $\mathbf{W}_{rec}$ are the input, output and internal weights respectively. f is the relu activation function and is defined as $f(x) = \max(0, x)$.

Each output of the network is a linear combination of the node activities and can be written as follows,

$$\mathbf{z} = \mathbf{W}_{out}\mathbf{r}$$

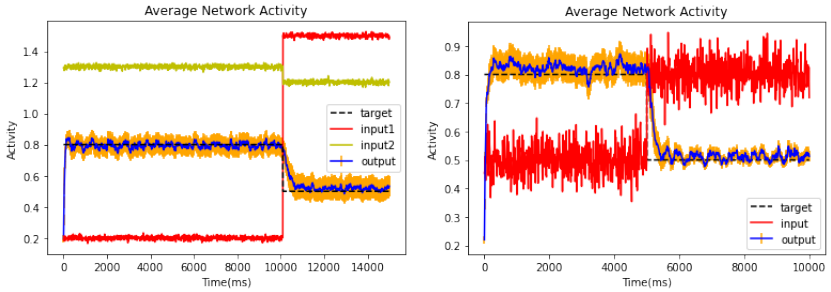


Figure 1: Both networks have low error. (Left) Averaged network output for the decision task changes from high to low when the relative difference between the inputs changes from negative to positive. (Right) Averaged network output matches the opposite condition of the input, as expected for the perception task.

Using Tensorflow, we simulated the first-order Euler approximation of the equation above with a timestep of $\Delta t = 10$ms. The discretized equation was,

$$\mathbf{r}_t = (1-\alpha)\mathbf{r}_{t-1} + \alpha f(\mathbf{W}_{rec}\mathbf{r}_{t-1} + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b} + \sqrt{2\alpha^{-1}\sigma^2}\mathbf{N}(0,1))$$

Where $\alpha = \Delta t/\tau$.

Internal weights were randomly initialized using a Gaussian distribution with mean 0 and variance $1/\sqrt{N}$ where N is the number of nodes in the network. The diagonal of the internal weight matrix was set to 0 so there were no recurrent connections. Output weights were chosen from the same distribution. Input weights and bias were similarly chosen by Gaussian distributions with standard deviations of $1/\sqrt{3}$ and $1/2$ for the perception and decision tasks respectively. Only the internal weight matrix was trained, but its diagonal values were maintained at 0.

The network was implemented using Tensorflow and trained using the Adam optimizer (gradient descent) with the default learning rate of 0.001. We used a l2 loss function over time for training.

3 Network Performance and Task Descriptions

The perception task asks the network to discriminate react in a simple way to a given fixed point, but noisy input. An input of of 0.8 should result in a network output of 0.5 and an input of 0.5 should result in an output of 0.8.

The decision task asks the network to examine two fixed point inputs and identify which of these inputs is larger than the other.

As we see in figure 1, all ten networks correctly perform the task assigned to them with low error.

4 Latent Space Visualizations

Here, we can visualize the network trajectories in a latent space while it performs the two tasks.

In the latent space trajectories for the decision task (Fig. 2), we see the fixed points are split into two layers, one where input 1 is greater and the other where input 2 is greater. The dynamics of the network connect these two layers as the input conditions change. The network performs the computation by going into one of the layers for one condition, and the other layer for the other condition.

We see that the initial transient dynamics for the perception task is orthogonal to the plane of both fixed points in the trained network, but is on the same plane for the untrained network (Fig. 3). This implies that the latent space of the trained network is rotated as to best identify the relevant points in the 0.8 to 0.5 input regime while the initial dynamics, orthogonal to this plane, are not as important.

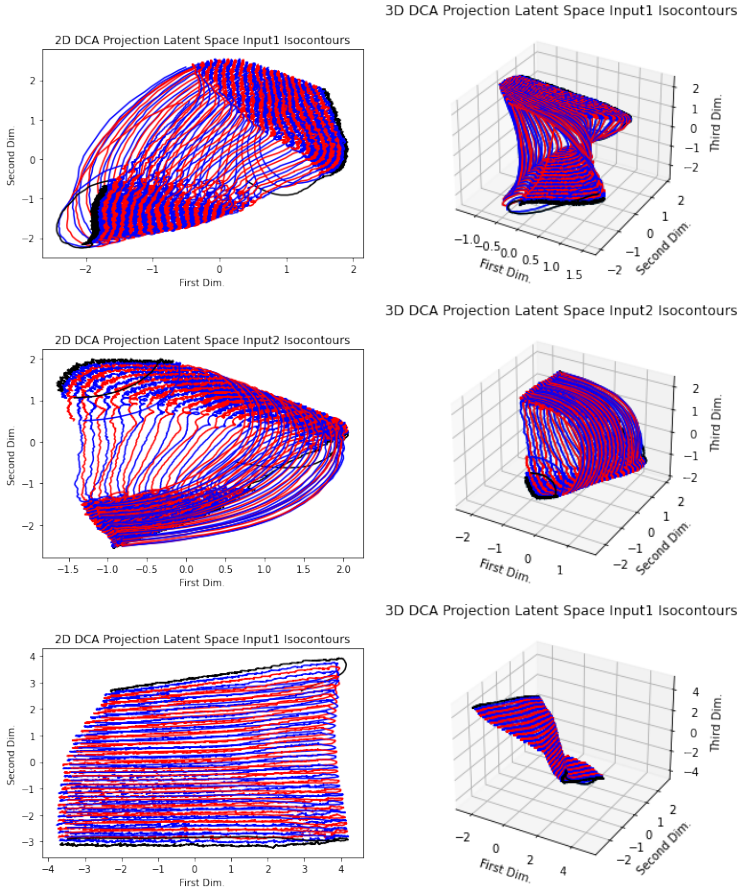


Figure 2: Latent space trajectories for the decision task. The alternating blue and red lines reflect the isocontours where one input was changing and the other was held fixed. The black lines represent the isocontours where the fixed input takes on its smallest or largest value. Note that there is some variability when the projection was repeated, but the major features are always there. (Top Row) The Input1 isocontours for the trained network. (Middle Row) The input2 isocontours for the trained network. (Third Row) Input1 isocontours for the initial, untrained network with random weights.

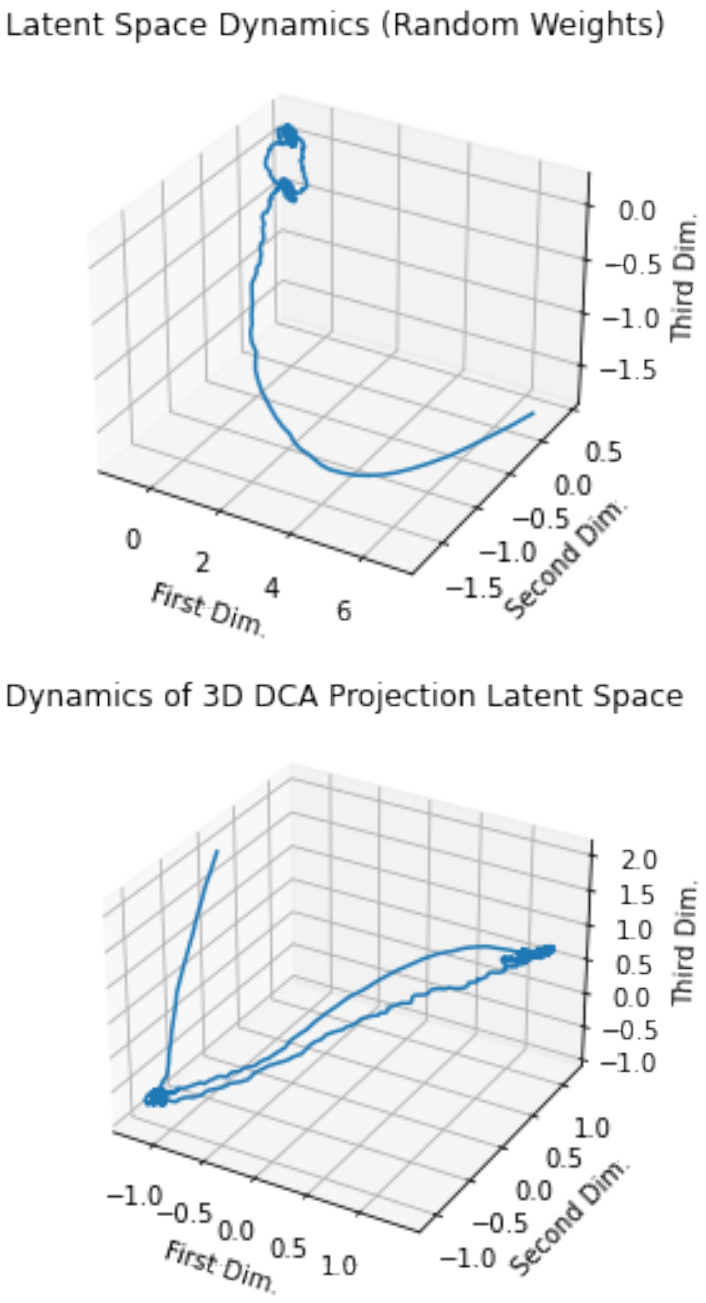


Figure 3: Perception Task latent space projection. (Top) Latent space trajectory for an untrained network with random weights and input values switching between 0.5 and 0.8. (Bottom) Latent space trajectory for a trained network with random weights and input values switching between 0.5 and 0.8.

5 Hypotheses

5.1 Increasing Nonlinearity

Here, we hypothesize that the network takes advantage of the nonlinearity to perform the tasks assigned to it. This hypothesis stems from the fact that the network dynamics are more nonlinear post-training than pre-training, and the network weights become negative on average during the training process which implies greater activation of the RELU nonlinearity.

In figure 4, we see that the weights decrease during the training process on average. Moreover, we see that this decrease makes their mean value negative. In addition, the decision task, which is arguably more complex than the perception task, has a larger mean weight decrease. This points to the idea that the networks are taking advantage of the RELU non-linearity to perform the perception task as well as the more complex decision task.

We applied DCA to get a measure of how linearly predictable future dynamics are from the past for the trained and untrained conditions. We varied the nonlinearity of the network by using a leaky RELU activation function (Fig. 5) and varying the angle between the function and the x-axis (θ). At $\theta = 0$, the activation function was equal to a RELU, and at $\theta = \pi/4$, the activation function was equivalent to that of a linear system. Surprisingly, we found that for both tasks, the trained maximal predictive information (PI) was lower than the untrained PI at $\theta = 0$, but as θ increased, the trained PI became greater than or equal to the untrained value (Fig. 6). Since the PI can be thought of as a measure of how linearly predictable future dynamics are from the past, this shift shows that the trained network dynamics are more nonlinear than the untrained dynamics.

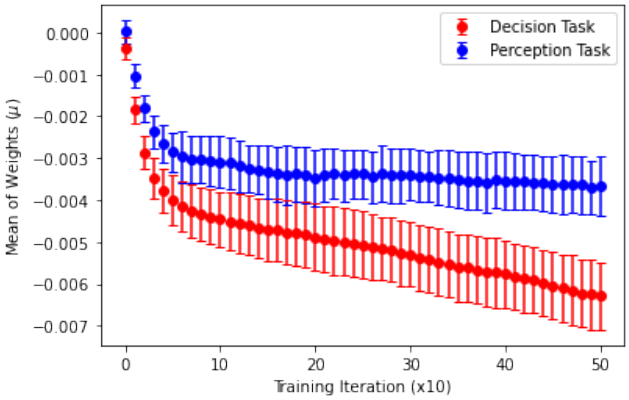


Figure 4: Average weights of the network decreases during training.

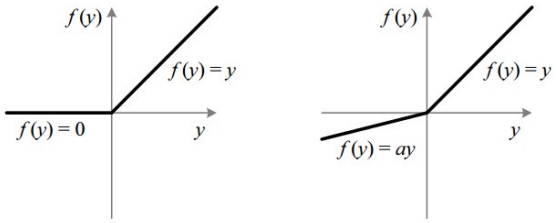


Figure 5: The leakyRELU activation function.

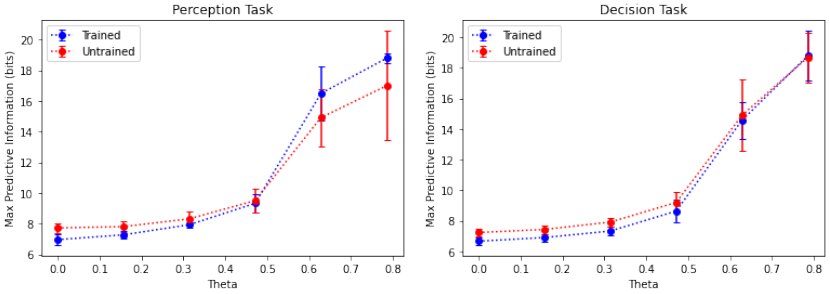


Figure 6: How the predictive information changes for the trained and untrained networks as the nonlinearity is decreased. (Left) For the perception task, we find that the trained predictive information overtakes that of the untrained network as the dynamics become more linear. (Right) For the decision task, we find that the trained predictive information becomes equal to that of the untrained network as the dynamics become more linear.

5.2 Subtle Changes Have a Big Impact

This 2021 paper by Schuessler, et. al (https://arxiv.org/pdf/2006.11036.pdf) shows that the weight changes during the training process are of low rank and tends to be relatively small. We see the same result in our networks, where there are few weights and eigenvalues of the weight matrix that have been changed (Fig. 7), and at first glance the changes do not seem significant. However, when examining measures such as node centrality and the maximally predictive latent space, we find that these few weight changes actually make a significant difference in computationally relevant metrics.

Surprisingly, these small changes in the weight matrix cause large differences in the centralities of the different nodes. In fact, we find that, on average, nodes with low initial centrality become the most important post-training and vice versa (Fig. 8). While the training process maintains most of the existing structure in the random network, it changes the "most important" pieces of that network.

We additionally see an increase in the non-normality of the weight matrix measured through its Schur decomposition. I omitted the plot, but the 2-norm of the upper triangular matrix in the decomposition (minus diagonal elements) changes from about 16.5 to 17.5 throughout the training process for both tasks.

Finally, we find that training shifts maximally predictive subspaces for networks trained for both tasks. In fact, the latent spaces of the relevant dynamics pre and post training are almost orthogonal in the lower dimensions (Fig. 9).

However, subspace angles between the latent space projections for the two different input conditions were almost the same pre and post training (Fig. 10). This implies the networks shifted both relevant latent spaces but also kept them fairly aligned.

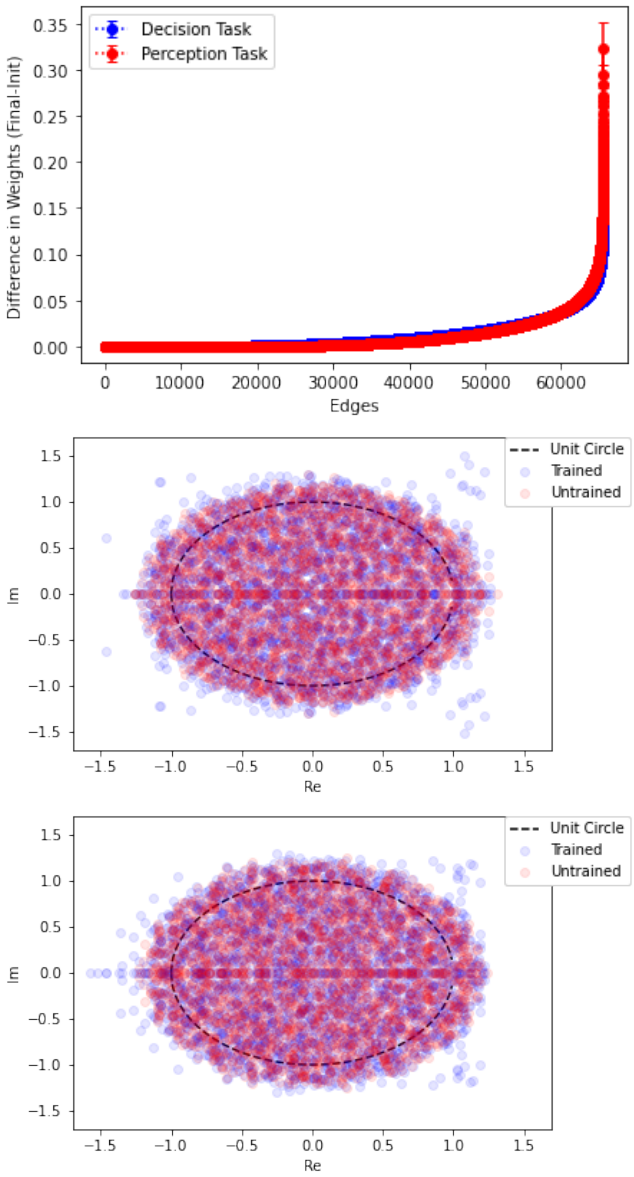


Figure 7: (Top) Only a few of the weights undergo significant changes and most are left unchanged during the training process. (Middle: perception, Bottom: decision) Similarly, most eigenvalues are left unchanged during training. A few have significantly increased magnitudes post-training.

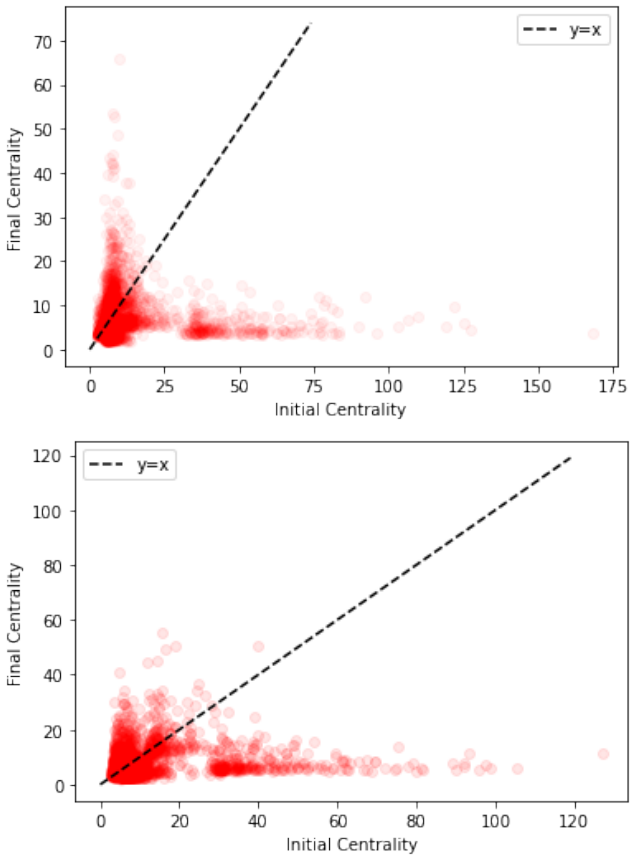


Figure 8: (Top) A scatter of the initial vs final centralities for each node in the networks trained for the decision task. (Bottom) Same thing but for the perception task. The 'L' shape in both plots implies the training process switches the most important and least important nodes. Note that the changes in these plots look more extreme than the average changes an individual network goes through. The centralities of a single network tend to be more localized than the plots suggest.

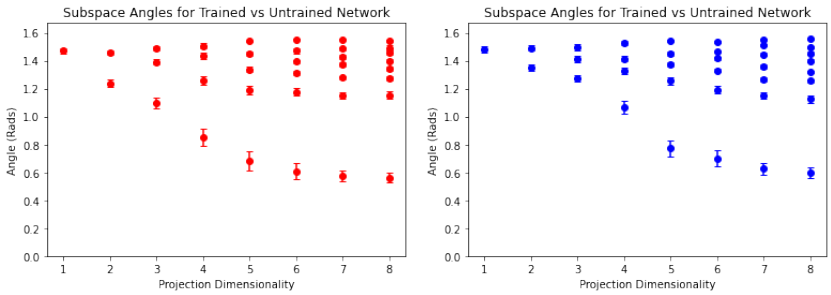


Figure 9: Subspace angles between maximally predictive latent spaces pre vs post training. (Left) Decision Task (Right) Perception Task

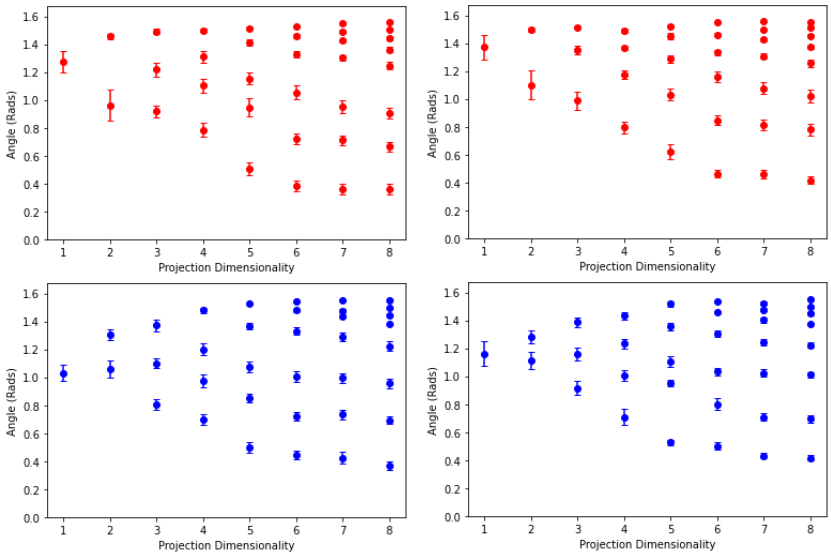


Figure 10: (Top) Decision Task (Bottom) Perception Task (Left) Trained Networks (Right) Untrained Networks. Plots of the subspace angles between maximally predictive latent spaces for both input conditions. In the trained networks, the latent spaces for the two different input conditions seem slightly more nested than in the untrained networks, but there is little to no change otherwise.

5.3 Latent Space Moves Away From Readout

In addition to the other changes made by the training process in the previous section, we find that the training process shifts the relevant latent subspaces so they are more orthogonal to the redout vector (Fig. 11). This might imply that important dynamical computations are shifted to trajectories in subspaces orthogonal to the readout to maximize the stability of the output. It is also apparent that the (arguably more relevant) DCA latent spaces are shifted further away from the redout those of PCA, which also supports the above claim (Fig. 11).

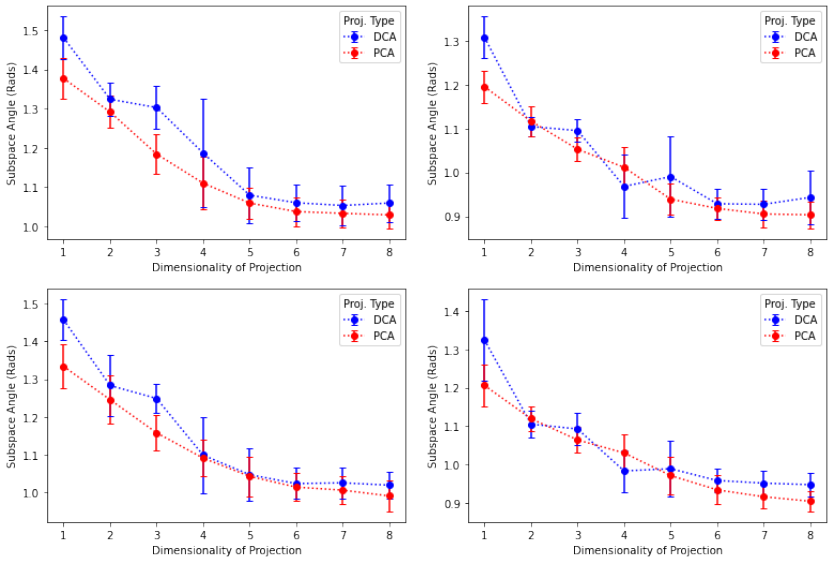


Figure 11: (Left: trained networks, Right: untrained networks; Top: decision task, Bottom: perception task) Subspace angles between the latent spaces of the different networks and the readout vector. The trained network latent spaces are more orthogonal to the readout than those of the untrained networks. Note: The axes above are not the same (sorry!)

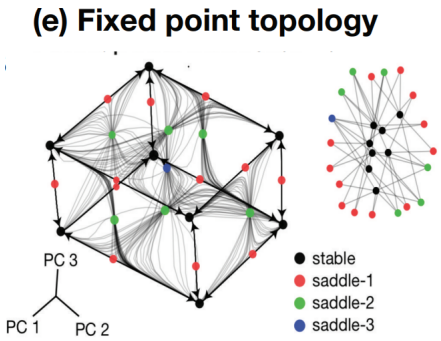
6 Future Work

In the future, we can further explore all of these hypotheses with the inclusion of sparsity regularizers, and varying task complexity. And also doing these analyses with many more networks (possibly on a supercomputer) to get more accurate results.

We propose two ideas to increase task complexity. The first is to increase the dimensionality of the fixed points. For example, for the perception task, the inputs could represent coordinates in some higher-dimensional space and the output can be the distance of these coordinates from the origin. For the decision task, we have a similar input of two different coordinates, and the network is asked distinguish which coordinate is farther from the origin.

The second idea to increase task complexity is to ask the network to do tasks with various dynamics. For example, outputting the frequency of a sine wave input, distinguishing which of two sinusoidal waves has a higher frequency, and/or outputting a sine wave with a frequency equal to a fixed (or changing) input.

In Sussilo and Barak, a method for numerically finding "slow points" in the network dynamics is proposed. Essentially, these are all fixed points and/or saddle points found from zeros of the Jacobian of the network dynamics. In Ganguli and Sussillo, this technique is used to visualize the phase space trajectories of various trained RNN architectures. Example:



The study concludes that across different architectural choices of the RNN, the number and characteristic (fixed vs. saddle) of slow points is fairly universal. Thus in both our work (see Figures 2/3) and here, the relevant trajectories for computation are found in the vicinity of a small number of saddles/fixed points. This raises a series of questions:

1. For tasks of the same complexity, are slow regions (collections of fixed points/saddles) re-used for different tasks, and simply modulated by external inputs? Or, do distinct tasks require largely non-overlapping slow regions?

2. As task complexity is increased, do the number of saddles/fixed points increase? At what rate?

3. Do inputs to the network modulate the slow point hessians of the autonomous networks?

4. Given some sense of the number of slow regions required to solve a task set, can one more easily train networks by initializing with connectivity matrices that already have slow points comparable to the final structure?

5. Can we define a notion of task complexity by linearly approximating the dynamics around the various fixed/slow points of a network while executing the task, and measuring predictive information in those regions? Would such a measure give consistent results for the same task executed by different networks?

6. Is reversibility decreased around slow points? i.e. do they funnel many trajectories into one?

7. What can we learn by analyzing linear models that are fit to the 'slow' dynamics?

Related to these questions is the seminal work of Wainrib and Touboul, where the number of slow points in an i.i.d Gaussian recurrent neural network is shown to scale exponentially with dimension in the vicinity of the transition to chaos. Thus, the presence of deterministic chaos (or equivalently, positive Lyapunov exponents), can be understood to result from the proliferation of saddle points leading to itinerant trajectories.

Reservoir computing, in which the readout of the network dynamics is optimized rather than the network weight matrix itself, harnesses the chaotic phase, but this is decidedly not the regime in which we operate. Indeed, our networks are initialized in the stable fixed point regime, and remain within this region after training. Importantly, inputs to the system can dramatically expand the region of stability of the stable fixed point phase. Therefore, a relevant question for identifying large capacity RNNs seems to be, for what network architectures do slow/fixed points proliferate *without* chaotic dynamics?

6.1 Implementing Slow Point Analysis

To find the slow points of the system, we need to find the regions of dynamics where the kinetic energy is at a "minima".

The dynamical equations of our recurrent neural networks are,

$$\dot{\mathbf{r}} = \frac{1}{\tau}[-\mathbf{r} + f(\mathbf{W}_{rec}\mathbf{r} + \mathbf{W}_{in}\mathbf{u} + \mathbf{b} + \sqrt{2\tau\sigma^2}\mathbf{N}(0,1))]$$

To understand the different terms in the dynamical equation as well as how it was discretized, see the Network Training Section. We define $\mathbf{F}(\mathbf{r})$ to be the RHS of the above equation.

$$\mathbf{F}(\mathbf{r}) = \frac{1}{\tau}[-\mathbf{r} + f(\mathbf{W}_{rec}\mathbf{r} + \mathbf{W}_{in}\mathbf{u} + \mathbf{b} + \sqrt{2\tau\sigma^2}\mathbf{N}(0,1))]$$

And in this case, f is the ReLU activation function. The kinetic energy is defined as,

$$q = \frac{1}{2}|\mathbf{F}(\mathbf{r})|^2$$

To find the slow and fixed points, differentiate the kinetic energy with respect to each r_i.

$$\frac{\partial q}{\partial r_i} = \sum_{k=1}^{N} \frac{\partial F_k}{\partial r_i} \dot{r}_k$$

6.1.1 Computing Fixed Points

A fixed point is defined as the point where $\dot{\mathbf{r}} = 0$, which also implies the kinetic energy is at a minima. We can estimate the fixed points from the dynamics of the system by computing the points where,

$$\dot{r}_k \approx \frac{\Delta r_k}{\Delta t} = \frac{1}{\tau}\Delta r_k \approx 0$$

Where the tolerance for equality with 0 should be on the order of the noise added to the system.

6.1.2 Computing General Slow Points

For more general slow points, we want

$$\frac{\partial q}{\partial r_i} \approx 0$$

for all i. Where, the tolerance for equality is once again on the order of the noise added to the system. We also want the matrix of second order derivatives of q to be positive definite. To compute both these values, some additional formulas are required.

We know that,

$$\frac{\partial q}{\partial r_i} = \sum_{k=1}^{N} \frac{\partial F_k}{\partial r_i} \dot{r}_k$$

and we know how to compute $\dot{r}_k$ in our discretized RNN approximation from the previous section on fixed points. All we have left to do is to compute $\frac{\partial F_k}{\partial r_i}$.

We know that,

$$F_k(\mathbf{r}) = \frac{1}{\tau}\Big[- r_k + f(m_k)\Big]$$

Where,

$$m_k = \sum_j W^{rec}_{kj} r_j + \sum_j W^{in}_{kj} u_j + b_k + \sqrt{2\tau\sigma^2} N(0,1)$$

The RNN class being used to conduct these experiments can be modified to store the m_k values at each timestep. This modification may not be required to compute these values, but will probably make it much easier to do so (no need to rewrite code to evaluate m_k at the different timesteps since the RNN class already did the computation during the simulations).

Letting f be the ReLU activation function, we find

$$\frac{\partial F_k}{\partial r_i} = \frac{1}{\tau} \begin{cases} \delta_{ik} & \text{for } m_k \leq 0 \\ \delta_{ik} + W^{rec}_{ki} & \text{for } m_k > 0 \end{cases}$$

And

$$\frac{\partial^2 F_k}{\partial r_i \partial r_j} = 0$$

Putting everything together, we can compute the general slow points by checking the equation,

$$\frac{\partial q}{\partial r_i} = \sum_{k=1}^{N} \frac{\partial F_k}{\partial r_i} \dot{r}_k \approx 0$$

at every timestep of the simulation. And by ensuring the second order derivative of the kinetic energy is positive definite. i.e. the matrix,

$$\frac{\partial^2 q}{\partial r_i \partial r_j} = \sum_{k=1}^{N} \frac{\partial F_k}{\partial r_i} \frac{\partial F_k}{\partial r_j} + \sum_{k=1}^{N} \frac{\partial^2 F_k}{\partial r_i \partial r_j} \dot{r}_k$$

must be positive definite for there to be a minima at a given timestep.

Then, one can linearize the dynamics of the network around these points for further analysis.