DIMENSIONAL REDUCTION WITH RESERVOIR NETWORKS

MIKE SCHACHTER

1. Introduction

Feedforward neural networks have the theoretical ability to be universal approximators, meaning they can be constructed to approximate any function comprised of input data and output data [4]. Given some data, we "train" a feedforward network by algorithmically tuning the weights that dictate the strength of connections between neurons until the network produces a good fit to the data. With the recent progress in unsupervised pre-training of individual layers of many-layered feedforward networks[2] ("deep nets"), it has become even more tangible to train complex networks that capture important variations in high-dimensional data.

However, there are computational concerns when the input data varies in time. For example, say we want to train a feedforward network to classify whether a spoken word was "cat" or "dog". Typically we would start by transforming the sound pressure waveform into a different representation called a spectrogram, which represents the frequencies in the sound that change over time. The spectrogram could be at least a 60 dimensional time series, i.e. some waveform $u(t) \in \mathbb{R}^{60}$. If we sample the frequencies every millisecond, and a spoken word takes at least one second, we would produce input samples that have a dimension of 1000*60=60,000. Although training on input with this dimensionality is achievable, we would like to investigate networks which have an intrinsic capability to compress a temporal sequence and store it. These compressed "memories" would ideally be used for complex temporal computations while removing the requirement to store so much data.

Recurrent neural networks are neural networks that have loops in their connectivity. They are directed cyclic graphical models. As a result, they can pass around information from previous time points. Their activation patterns at each time point represent not just the instantaneous stimulus $u(t)$, but information from previous states of the stimulus $u(t-1), ..., u(t-N)$. For reasons that we will not get into, there are some problems with training the weights of recurrent networks; they can be a bit finicky.

One way recurrent nets are constructed and used falls under the recent moniker of "Reservoir Computing". Reservoir computing requires the construction of a reservoir, a recurrent dynamical system such as a recurrent neural network that has the capacity to process and store temporal patterns. To construct these networks for a classification task, first we generate a random, sparsely connected recurrent network whose weights are within a certain range. Then we run the input through the network, and record the state of the network after an input has been presented. Finally, we train a simple readout classifier that predicts the stimulus class from that network state [5, 8].

In this study, we construct a class of small, fully-connected recurrent networks with random weights that perform dimensionality reduction on temporal input patterns, i.e. they store a compressed "memory" of the input that can be read out and used to classify the temporal pattern. We quantify the quality of the dimensionality reduction by examining the performance of readout classifiers, and attempt to figure out what structural features of the weight matrix dictate the performance of a readout classifier.

2. Methods

All source code for this project is at http://github.com/mschachter/prorn.

2.1. **How the Networks are Constructed.**

We use an 3-node recurrent network with linear units as shown in figure 2.1. The state of the network at time $t \in \mathbb{Z}$ is a vector $x(t) \in \mathbb{R}^3$. The stimulus is a time series $u(t) \in \mathbb{R}^M$, where in this study $M = 1$. More on the stimulus below. The weights between nodes are stored in an NxN matrix $W = (w_{ij})$, where w_{ij} is the weight of the directed edge from i to j. The weights between elements of the stimulus and the network are stored in an MxN matrix $W^{in} = (w^{in}_{ij})$, where w^{in}_{ij} is the weight of the directed edge from input $u_i(t)$ to node j. The update equation of the network is:

$$x(t+1) = Wx(t) + W_{in}^T u(t) \tag{2.1}$$

The weight matricies are initialized to be independent Gaussian random numbers from $\mathcal{N}(0,1)$. They are then divided by αw_{max}, where w_{max} is the maximum weight and $\alpha \in (0,1]$ dictates how close the absolute values of a weight gets to 1. When α is set close to 1, the network operates close to the "edge of chaos" [7]. This is related to how the eigenvalues of the weight matrix W in difference equation 2.1 dictate whether the network will settle down into a stable attractor state in the absense of stimuli. Let $\lambda_i \in \mathbb{C}$ be an eigenvalue of W, when all $|\lambda_i| < 1$, the system is said to be "stable" and will settle down in the absence of input. If any $|\lambda_i| = 1$, the system is "neutrally stable", and when at least one $|\lambda_i| > 1$, the system is "unstable" and will diverge into a chaotic state [9].

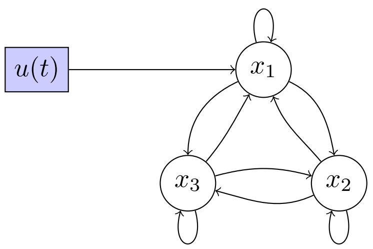

Figure 2.1. The recurrent network used in this study.

We constructed approximately 4500 random networks, where each connection weight was randomly generated from $\mathcal{N}(0,1)$, and then rescaled as noted above with $\alpha = 0.99$. The weight matrix W was $3x3$, figure 2.2 shows histograms of $|\lambda_1|, |\lambda_2|, |\lambda_3|$ across all networks constructed. The eigenvalues are ordered by size, i.e. $\lambda_1 = \lambda_{max}$ for all networks. Note that $|\lambda_1| > 1$ for some networks.

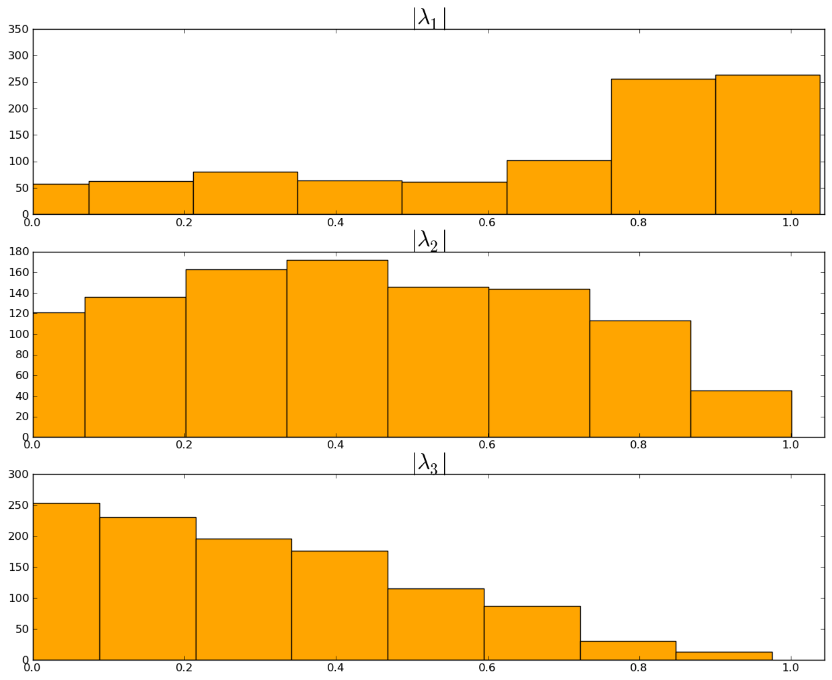

FIGURE 2.2. The weight matrix eigenvalue distributions across all constructed networks.

2.2. **Stimulus Construction.** The stimuli used in this study are shown in figure 2.3. They are finite with length $T_u = 15$, and constructed as follows:

(1) Randomly choose the number of stimulus "bumps" N_b, from a uniform distribution of integers in $(1, \gamma T_u)$, where $\gamma \in (0, 1)$ sets the maximum number of bumps.
(2) Generate each bump $b_i = (t_i, h_i, \sigma_i)$:
    (a) Choose a bump time, a uniform random integer $t_i \in (0, T_u)$ that has not been chosen yet.
    (b) Choose a bump height, a uniform random integer $h_i \in (1, h_{max})$, where $h_{max} \in \mathbb{N}$ determines the number of discrete "amplitudes" that are possible in the stimulus.
    (c) Choose a random standard deviation $\sigma_i \sim \mathcal{N}(0, \sigma_{avg})$, the bump spread.
(3) Generate the stimulus prototype as:

$$u(t; b) = \sum_{i=1}^{N_b} h_i exp \left\{ \frac{(t - t_i)^2}{2\pi\sigma_i^2} \right\}$$

(4) Generate a family of stimuli from the prototype by adding Gaussian noise.

The stimulus was always attached to node x_1. We used 5 different stimulus prototypes, and from these 5 prototypes we generated 500 noisy sample stimuli for each class, according to the above algorithm. Again, figure 2.3 illustrates these stimuli.

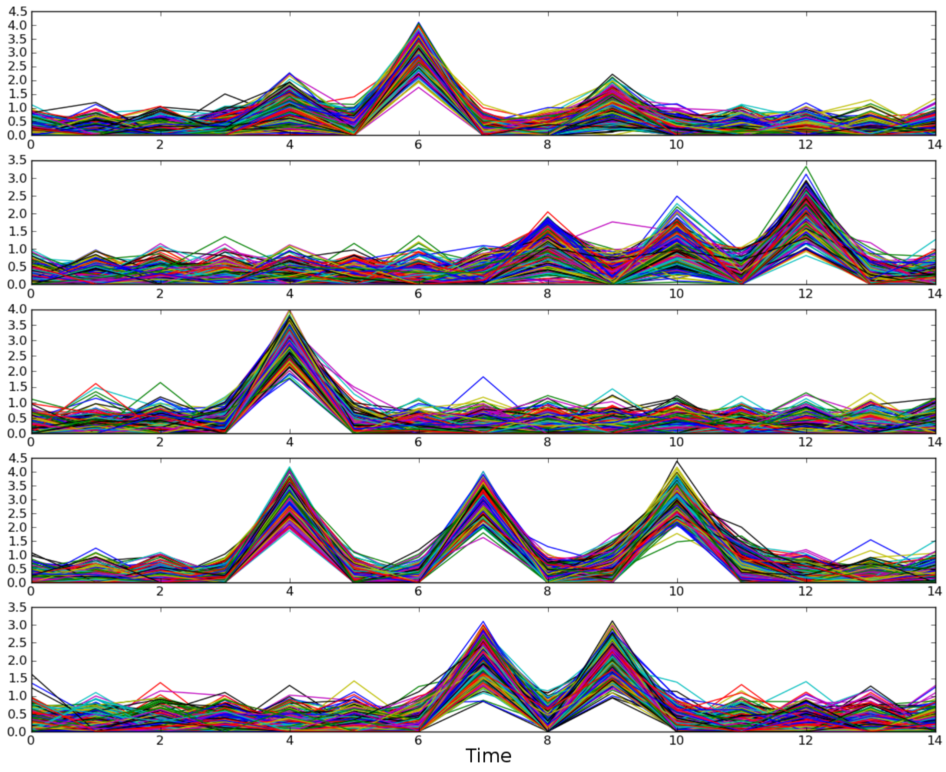

FIGURE 2.3. Stimulus patterns used as input to network.

2.3. **Readout Classifier Construction.** Each stimulus prototype described above is used to generate a family of temporal patterns in one of five classes. We want to present the stimulus to the recurrent network, measure the instantaneous state following stimulus presentation, and then train our readout classifier to predict the class of that stimulus based on the state.

Given a network, we generate the samples for a classifier as follows. First, we let the network equilibrate to a rest state in the absence of input, over 500 time steps. As noted above and shown through simulation, any network with $|\lambda_{max}| < 1$ will exhibit this behavior. We label the equilibration time period as $-500 \leq t < 0$. At $t = 0$, we switch on a stimulus of class $\mathcal{C}_i$, which runs until time $t = T_u - 1$. We then record the network state $x(\tau)$, where $\tau = T_u$, and create a sample $(x(\tau), \mathcal{C}_i)$ comprised of the state and stimulus class. With 500 noisy stimuli per class, and 5 classes, we obtained 2500 training samples per network in a dataset $\mathcal{D} = \{(x(\tau), \mathcal{C}_i)\}$. 75% of the data was used for training and 25% was held out for validation.

We tried two different readout classifiers. The first utilized logistic regression in a one-vs-all configuration. Given one class $\mathcal{C}_i$, we labeled samples belonging to $\mathcal{C}_i$ as 1, and the rest of the samples to 0. We then trained a logistic regression to predict membership in $\mathcal{C}_i$ or not. The performance was quantified per-class with percent correct. Our overall performance measure for the logistic regression readout is the average percent correct across classes.

The second classifier was a two-layer feedforward neural net, with 2 hidden nodes and 5 output nodes. The middle nodes utilized a tanh activation function, and the output nodes utilized a softmax activaction function. By doing so, the network learned to predict the posterior probabilities $p(\mathcal{C}_i|x)$. However, we forwent an extensive exploration of neural network performance measures and relied on the crude percent correct on the test set as the overall performance measure for the network readout. The two classifiers had somewhat proportional performances, as seen in figure 2.4. The neural network performance was much more variable than for logistic regression. Lack of computational power prevented us from using multiple initial guesses and averaging performances to get an overall performance number.

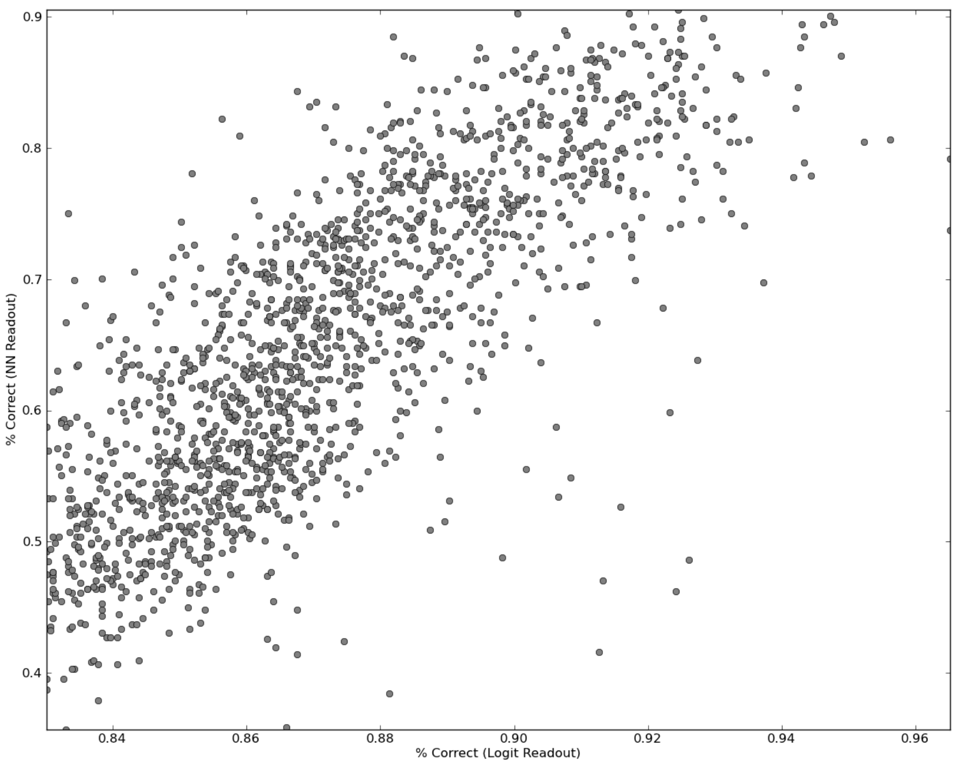

FIGURE 2.4. Relationship between performance of logistic regression and feedforward neural network readouts used to classify a stimulus class based on network state, after observing the stimulus.

There are several reasons to try out at least two different classifiers. First, logistic regression can only handle linearly-separable classes, whereas a feedforward neural network is (in an ideal situation) a universal approximator that can deal with nonlinearly-separable classes [4]. Second, the one-vs-all (also known as "one-vs-rest") configuration of our logisitc regression implementation leads to regions of input space that are ambiguously classified [3].

One particular advantage of logistic regression over neural nets is that they are very quick to train. There is a reasonably proportional relationship between performances in the two types of readout implementations (figure 2.4), and if we were to "breed" recurrent networks using readout peformance as a fitness function, we could take solace in the fact that logistic readouts could be used to evaluate that fitness function quickly.

Regardless, the goal of this study is not to build a better readout, but to try and understand why the network provides a reasonable temporal kernel for the readouts, whatever implementation we use.

2.4. **A Information Theoretic Predictor for Readout Performance.** We would like to measure network performance in a model independent way, in the context of predicting how well a readout could possibly do. As mentioned later in the results, an ideal network exhibits states that are close together for stimuli of the same class, but farther away from states in other stimulus classes. We used mutual information in an attempt to quantify this effect.

Let $R = 5$ be the number of stimulus prototypes used and $\mathcal{C}_i$ be a stimulus class. For a given network, the dataset $\mathcal{D}$ is comprised of all recorded pairs $(x(\tau), \mathcal{C}_i)$, where $i \in \{1, 2, 3, 4, 5\}$. We want to look at two distributions; the distribution of network states at time τ given a particular $\mathcal{C}_i$, $\mathbb{P}[x(\tau)|\mathcal{C}_i]$, and the class-independent distribution of network states $\mathbb{P}[x(\tau)]$. We want to compute the entropy for both of these distributions. Since $x \in \mathbb{R}^3$, we bin the data in a 3-dimensional histogram, and compute the empirical entropies:

$$H[x(\tau)] = \sum_{x(\tau)\in\mathcal{D}} p(x(\tau))\, log_2 p(x(\tau))$$

and

$$\begin{aligned} H[x(\tau)|\{\mathcal{C}_i\}] &= \sum_{\{\mathcal{C}_i\}} p(\mathcal{C}_i) \sum_{x(\tau)\in\mathcal{D}} p(x(\tau)|\mathcal{C}_i)\, log_2 p(x(\tau)|\mathcal{C}_i) \\ &= \frac{1}{R} \sum_{\{\mathcal{C}_i\}} \sum_{x(\tau)\in\mathcal{D}} p(x(\tau)|\mathcal{C}_i)\, log_2 p(x(\tau)|\mathcal{C}_i) \end{aligned}$$

The mutual information is then given as:

$$\mathcal{I} = H[x(\tau)] - H[x(\tau)|\{\mathcal{C}_i\}]$$

We'll explain more about why this measure may be useful soon.

3. Results

3.1. **The Recurrent Network Performs Dimensionality Reduction.** The dimensionality of our stimlus is $T_u = 15$. It's reasonable to assume the stimuli live on a lower dimensional subspace, given the manner in which they're generated. Our recurrent network is a 3-dimensional entity. We have trained readout classifiers to predict the class of a 15-dimensional stimulus based on a single readout from a 3-dimensional network state. Intuitively, we can think of the recurrent network as performing a dimensionality reduction on the temporal stimulus. See figure 3.1 for some visual intuition.

A commonly used tool for dimensionality reduction is principle components analysis. PCA works by constructing a covariance matrix of the stimuli, taking it's eigenvectors, and using a subset of the eigenvectors that correspond to the largest eigenvalues to project the data into a lower dimensional subspace. For comparison with what the recurrent network is doing, we performed PCA on our stimulus set and projected the data into a 3-dimensional subspace. Figure 3.2 shows the results.

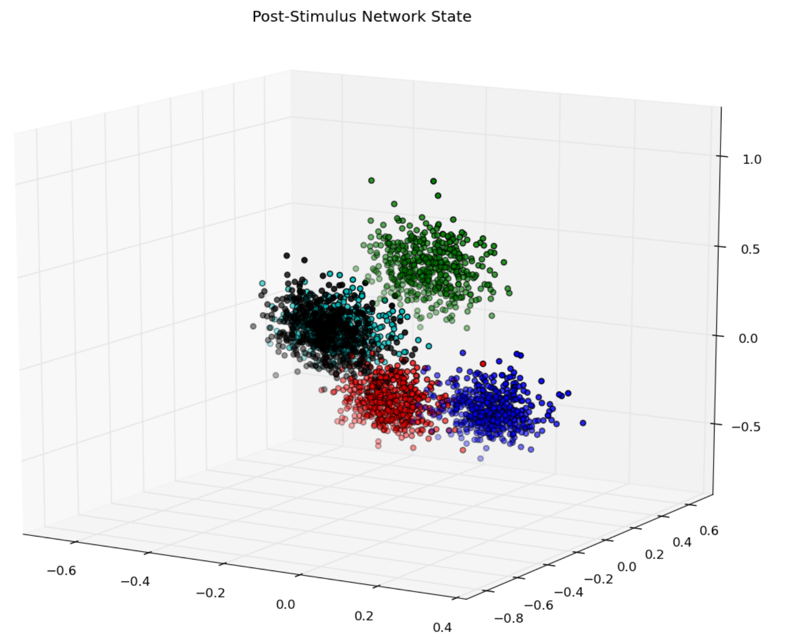

Figure 3.1. Each point is a network state recorded directly after the presentation of a stimulus, colored by the class of the stimulus. States from the same class lie in a cluster. In essense, the recurrent network is performing a dimensionality reduction on the temporal stimulus, projecting it into a 3-dimensional network state. Note that this particular network does not do a good job of separating the blank and cyan stimulus classes - others do better.

As mentioned in section 2.4, we attempted to quantify the separation of clusters using mutual information. For reference, the mutual information of network states in figure 3.1 was 1.4 bits, while the PCA projection in figure 3.2 was 2.3 bits. We hoped that, for network states that are well separated by stimulus class, the overall entropy $H[x(\tau)]$ would be high compared to the average entropy of each cluster $H[x(\tau)|\{\mathcal{C}_i\}]$, and the mutual information $\mathcal{I} = H[x(\tau)] - H[x(\tau)|\{\mathcal{C}_i\}]$ would be high.

Unfortunately, using a histogram to compute the empirical entropy of a 3-dimensional continuous variable is frought with issues. As noted in [1], histogram estimators for continuous entropy measurements can exhibit serious bias with non-univariate distributions. Due to some numerical issues that are still being worked through, we successfully estimated $H[x(\tau)|\{\mathcal{C}_i\}]$ for only half of the networks. The mutual information had a dependence on the histogram bin size used to estimate it, although not shown here we continued to increase the bin size until the entropies plateaued.

Technical problems aside, figure 3.3 shows that there is a relationship between readout performance and the mutual information $\mathcal{I}$ between the network state

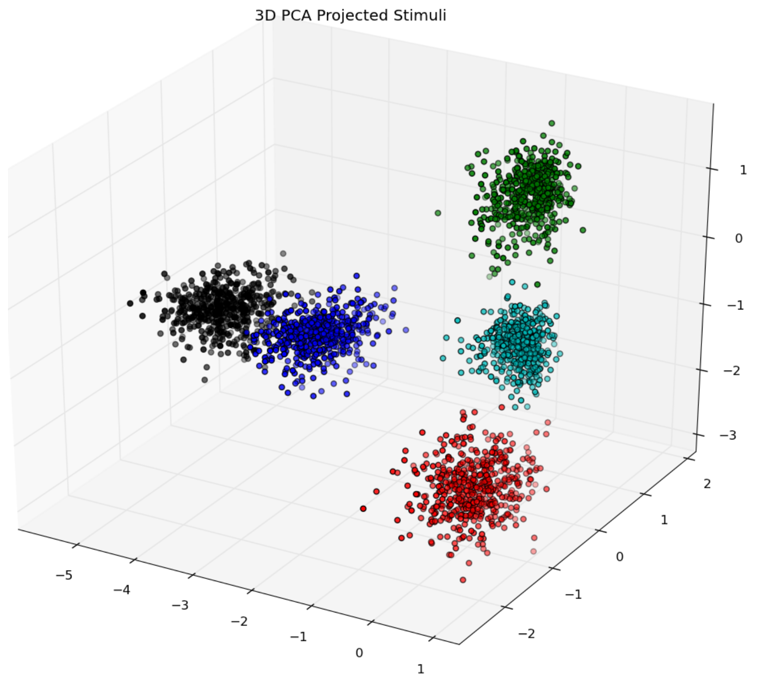

FIGURE 3.2. The stimuli in figure 2.3 projected onto a 3-dimensional subspace using PCA. Note the color scheme is the same as in figure 3.1; PCA projects blue and black classes closer together in 3D space, while the network in figure 3.1 projects black and cyan as close together. Although it's hard to see by comparing to figure 3.1, many recurrent nets projected the stimuli to a 2-dimensional manifold in a 3d state space, wheras PCA uses all 3 dimensions.

and the stimulus class. Although not sufficient, having a relatively high mutual information does seem to be a necessary condition for good readout performance.

3.2. **Recurrent Nets that Separate Stimulus Classes Well Exhibit Better Readout Performance.** One result that seems to have emerged from this study is that a recurrent network whose state separates stimuli well has better readout performance. Figure 3.4 shows the network state at readout time τ for the best performing network. It clearly separates different temporal stimuli, thus performing a good job of dimensionally reducing a higher dimensional stimulus set. In comparison, a two-layer neural net readout trained on the PCA data outperforms one trained on the state of this recurrent network by about 3-5% (not shown). However, PCA requires knowledge of the entire dataset to do it's dimensional reduction; while all the recurrent network has to do is exist! PCA also requires a projection of the high dimensional stimulus, which requires the stimulus to be stored in memory. There is no such constraint for the recurrent network.

3.3. **Relationship Between Eigenvalues, Mutual Information, and Performance.** There are relationships between how close recurrent networks come to the

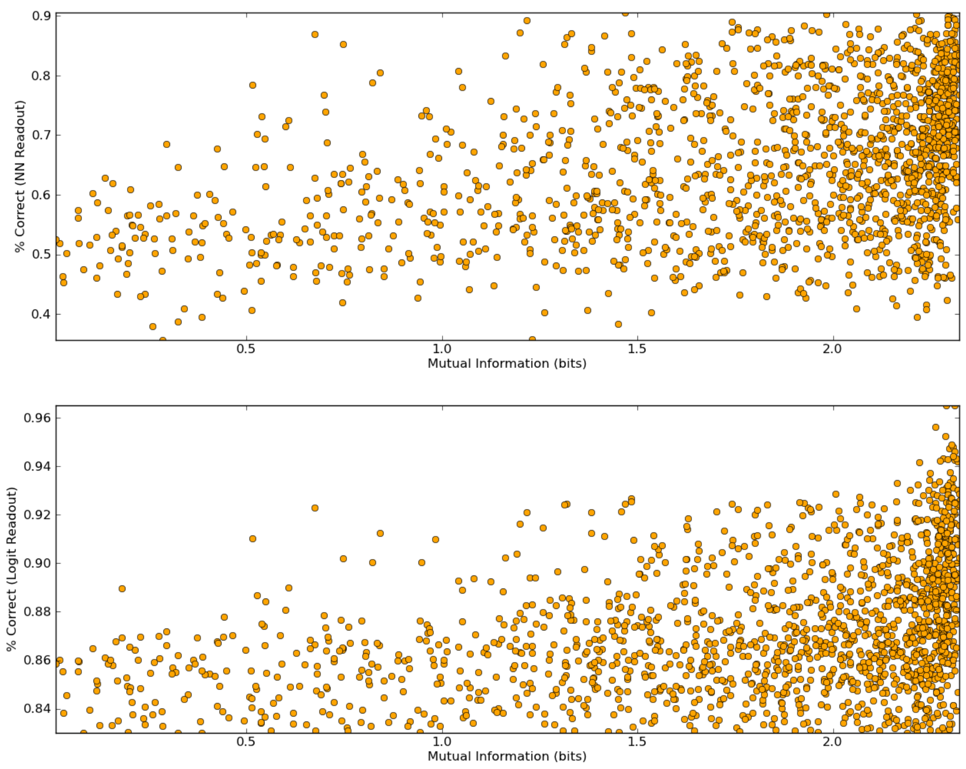

FIGURE 3.3. Readout performance as a function of mutual information between network state $x(\tau)$ and stimulus class $\mathcal{C}_i$. Lower performance means higher percent correct and a better readout.

"edge-of-chaos" and their ability to store memory traces in their states following stimulus presentation [7, 5]. Networks with a weight matrix whose absolute eigenvalues are close to 1 have a tendency to exhibit good properties for readouts. We have provided some intuition as to what these properties are; specifically showing that networks with good readout performance have a tendency to well-separate stimulus classes in their readout states.

Now we will jump the shark. Figure 3.5 is a 4-dimensional scatterplot, where each point is a recurrent network. The top two eigenvalues $|\lambda_1|$ and $|\lambda_2|$ are on the xy-axis, and mutual information is plotted on the z-axis. The network performance is colored, red is good, blue is bad. The figure clearly shows that the best networks for readouts are ones whose top eigenvalues are close to 1, and also have high mutual information. Future work will untangle the relationships between these properties and structural constraints on the network's weight matrix W.

4. DISCUSSION

This writeup documents a pilot study into the properties of a class of recurrent neural networks whose weights are not trained. The weights are randomly generated

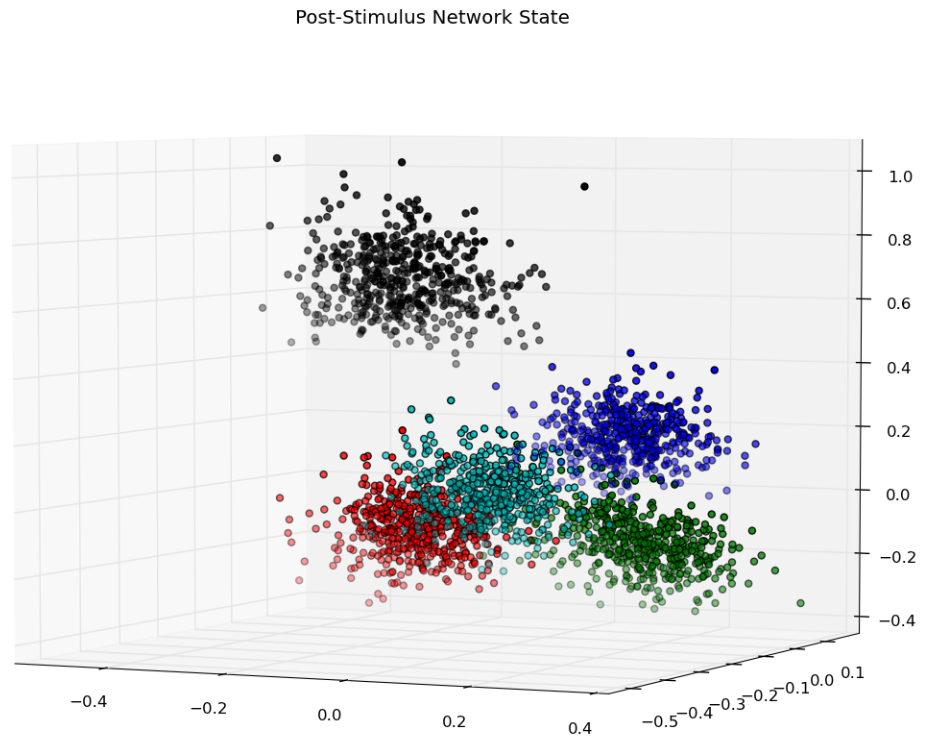

FIGURE 3.4. The recurrent network whose readouts performed the best exhibits well-separated clusters in it's state.

and static, and a simple linear readout is built to classify a temporal pattern after it is presented. There are many avenues for improvment..

Future directions include further study of recurrent networks that do dimensional reduction, but also networks whose state dimensionality is larger than the support for the temporal stimulus. Such networks are closer to what a support vector machine is - projecting to a higher dimensional, potentially nonlinear space in order to better separate the input patterns. Also, we are primarily interested in the study of the brain. In the brain, connection weights change as a function of the patterns of internal state, a phenomenon called synaptic plasticity. One group has begun studying the properties of recurrent networks with such plasticity [6], and we hope to move in that direction as well.

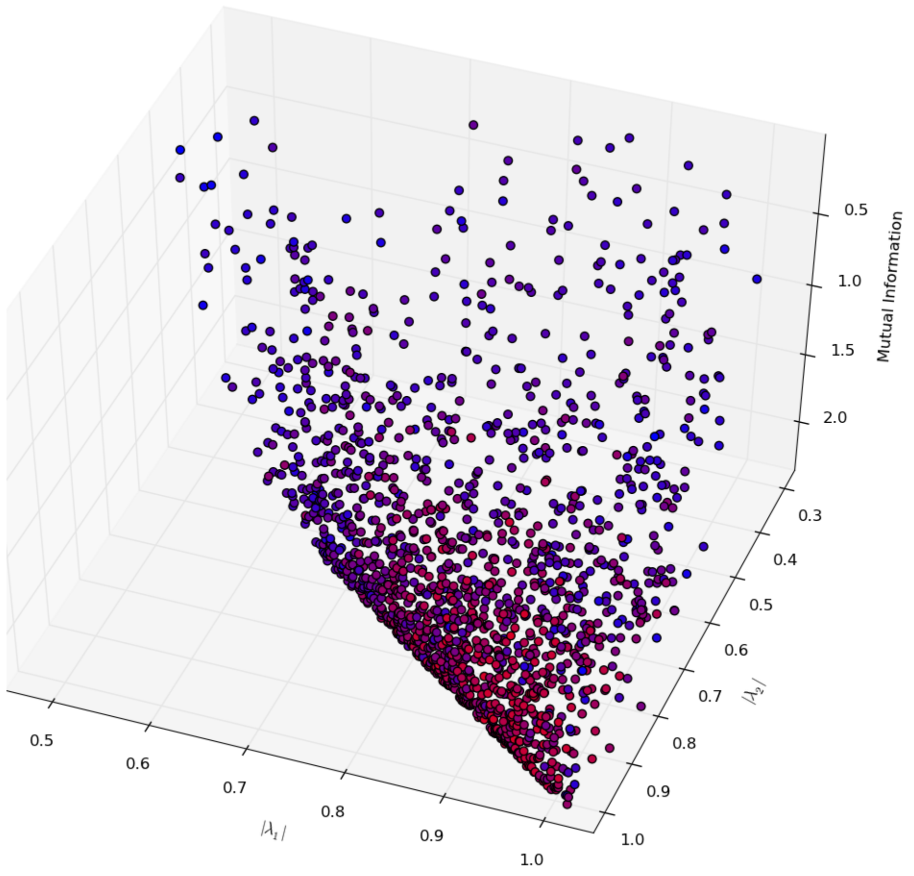

FIGURE 3.5. Redder is better - each point is a network; top two weight matrix eigenvalues $|\lambda_1|$ and $|\lambda_2|$ comprise the xy axis, and mutual information is the z axis. Readout performance is represented by the color of each point.

References

[1] J Beirlant, E J Dudewicz, and L Gyorfi and
E C van der Meulen. Nonparameteric entropy estimation: An overview. *International Journal of Mathematics and Statistical Science*, 6.1:17–39, 1997.

[2] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2.1:1–127, 2009.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] Kurt Hornik, Maxwell Sinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[5] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks. Technical report, Fraunhofer Institute for Autonomous Intelligent Systems, 2001.

[6] Andreea Lazar, Gordon Pipa, and Jochen Triesch. Fading memory and time series prediction in recurrent networks with different forms of plasticity. *Neural Networks*, 20:312–322, 2007.

[7] Robert Legenstein and Wolfgang Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20:323–334, 2007.

[8] Wolfgang Maass, Thomas Natschlager, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14:2531–2560, 2002.

[9] Gilbert Strang. *Linear Algebra and it's Applications (4th Edition)*. Thomson Brooks/Cole, 2006.