Dynamics of Recurrent Neural Network Models of Working Memory

Christian Wawrzonek

Advised by Timothy Buschman

Collaboration: Pavlos Kollias, Matthew Panichello

This paper represents my own work in accordance with University Regulations.

Department of Computer Science

Princeton University

April 29, 2016

Abstract

How do populations of neurons encode information over very short time scales? Given extensive training, neurons are able to change the weighted connections between them in order to encode information. However, very short timescales of only a few seconds are far too short to change neural weights. Still, humans and higher functioning animals possess the ability to encode and maintain small amounts of information presented over very short timescales. This is the problem of working memory, the transient holding, processing, and manipulation of information used in higher cognitive functioning. Previous computational models of working memory have typically been constructed with a strict, hand-tuned architecture. Here, we attempted to train a relatively simple, unconstrained neural network on complex working memory tasks and analyzed the natural solution space found by the network. Through a range of analyses, it is clear that even a simple, single layer recurrent network is capable of dynamic, generalized solutions without deliberate solution paths presented [8].

Acknowledgements

I would just like to take a moment to thank those who have helped me over the past year. Without their support and patience, I woundn't be here. Thank you to Matthew and Pavlos for being a huge source of guidance and inspiration throughout this year and last year. I have learned so much from both of you. I really think I'll miss our weekly meetings. Good luck with the research.

Thanks to Cindy for being supportive in those last few rough weeks. And of course, thanks to my roommates Mike and Ethan for putting up with all the late night antics.

Thanks to my best friend Peter, for pulling countless all nighters with me for no apparent reason and spending the whole night talking about philosophy, math and neuroscience.

Above all, thank you to the one and only Timothy Buschman. It has been an absolute honor to listen and learn from one you, undoubtedly one of the most brilliant people I've ever had the pleasure of working with.

1 Introduction and Background

1.1 Motivation and Goal

The motivation and style of this project follows very closely to the work of Mante et al. in their exploration of context dependent information integration in prefrontal cortex. In their words, "most often in computational studies in neuroscience a network model is designed by hand to implement a specific function, such as a decision, or auto completion of memories. This study was different [9]." Similar to Mante et al. we trained multiple networks to solve the same working memory task using an unconstrained, machine learning approach, and looked for similarities across successful models to determine natural solutions to the working memory problem.

Such unconstrained neural networks are often treated as a "black box" in machine learning. That is, free from the constraints of a deliberate structure designed to conform to a specific solution, neural networks such as the one in this project are treated as an opaque system with mysterious dynamics. However, given new techniques capable of breaking down a network, the underlying strategy or solution space can be understood directly. Sussillo and Barak call this process "opening the black box" [17]. This is the foundational problem of systems neuroscience. Understanding solutions in networks such as this is synonymous with breaking apart the dynamics of actual neural circuits to understand the underlying mechanisms implemented by the brain.

While Mante et al's project modeled an information integration task in PFC, the ultimate goal of this project is to directly model behavioral and neural data of working memory tasks, including LIP and FEF recordings. However, at this early stage, this was not a priority. As such, the results of this model cannot make strong claims about the actual neural mechanisms underlying similar behavior without further investigation. Timothy Buschman's lab is conducting research of very similar behavioral tasks, and future iterations or projects

similar to this would benefit from including more ties to empirical results.

1.2 Working Memory

Working Memory is the ability to hold and manipulate sensory information over very short timescales. It is an an essential component of cognitive control and attention. According to Ma et al, "It has been associated with persistent neural activity in many brain regions2 and is considered to be a core cognitive process that underpins a range of behaviors, from perception to problem solving and action control."

1.3 Delayed Saccade Task

A common task used to measure and model visual working memory is a delayed saccade task [1, 3, 5, 11]. A simple delayed saccade task is, as it is called, a task in which a subject, in most cases a monkey, must hold information about a stimulus in working memory for a short delay period and then execute a saccadic eye movement based on a remembered stimulus. The procedure of a delayed saccade task is illustrated in the figure below. The subject first fixates on a fixation point, usually in the center of a computer screen. Next, a location stimulus will appear in the periphery while the subject fixates on the fixation point. The location will then disappear and the subject must remain fixated on the fixation point until some cue time signals a saccade. This is usually cued by the disappearance of the fixation point or some additional stimulus. Finally, the subject must saccade to a location in space that corresponds to the relevant remembered stimulus. The right side of the Figure 1 shows 8 saccade directions that the monkey must remember and typical saccade responses around the target location, as presented in [14]. Our model extends this further to model anywhere between 4 and 32 locations.

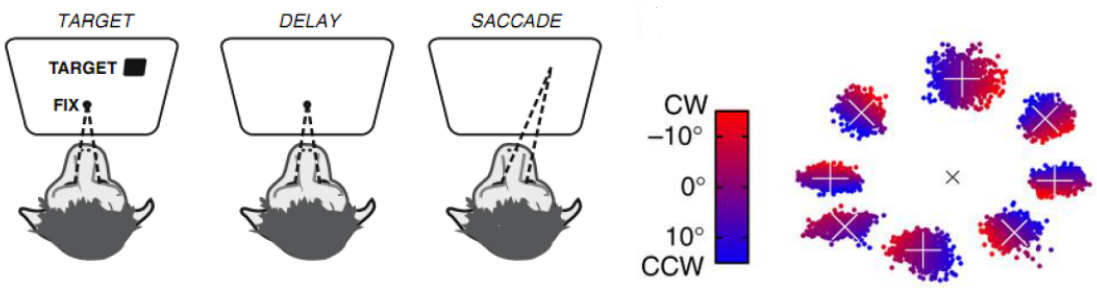


Figure 1: Basic design of the delayed saccade task as performed in Wimmer et al. [19]

Last year, my spring independent work consisted of engineering and studying the dynamics of a deep learning neural network modeling a simple delayed saccade task as described in Figure 1. This project significantly expands on the simple saccade task to a more complicated task that includes competing representations in working memory. Our current network models a variation of the simple saccade task with a combination of two similar working memory tasks, a selection task and an attention task.

In both tasks, there are two separate locations presented. In the figure, these are presented simultaneously. In our model, these are presented sequentially. Throughout this paper, the first location presented is referred to as sample location 1, while the second location is referred to as sample location 2. The location of the desired saccade is referred to as the target location. In addition, there is a selection parameter presented, either a 1 or a 2, that instructs the subject to saccade to either the first or the second sample location. In the attention task, the selection cue is presented before either sample location so the subject need only attend to the relevant sample location and ignore the unattended stimulus. In the selection task, the selection cue is not known until both locations are observed. In this case, the subject must maintain two competing location representations in working memory until the selection cue is known.

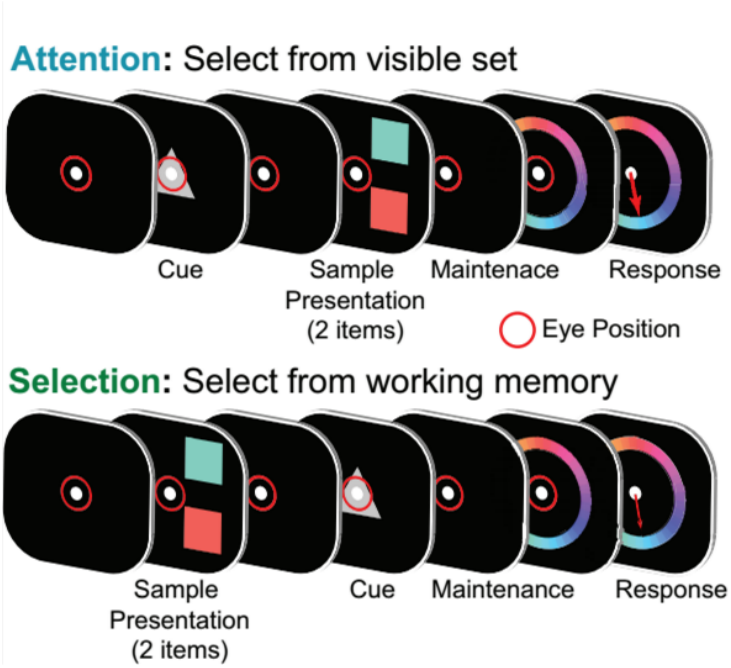


Figure 2: Design of the delayed saccade task modeled in this project. The top row shows the structure of an attention task in which the selection cue is presented before either location stimulus. The second row shows the structure of a selection task in which the locations are presented before the selection cue.

We created a multitude of networks that learned one of these two tasks. In addition, we created "combined" networks which are capable of performing both of these tasks simultaneously to see if such networks also created novel solutions given both paradigms, or simply combined the two solutions independently.

2 Model

2.1 Architecture

The task was modeled using a simple, single layer recurrent neural network architecture with with an input layer and output layer. The input layer was a single vector of $N + 4$ units where N was the total number of possible saccade locations for that task.

The 4 additional input units were used to represent the additional task information, namely the presence of the fixation point and the selection cue. The hidden layer is an all-to-all vector of recurrent units of varying length depending on the parameters of the model iteration. The output layer are 4 units, corresponding to the 4 directions on the Cartesian plane $(+dx, -dx, +dy, -dy)$. This projects the discrete input locations onto the unit circle. Rather than simply outputting a copy of the input vector, this transformation to Cartesian space necessitated the transformation of discrete inputs onto a continuous, circular space. That is, input location i is adjacent to $i + 1$, and therefore their representations should be continuous.

Figure 3 shows the architecture of the network while Figures 4, 5 and 6 show an example progression trial with inputs (Fig 4) and target outputs (Fig 5) of the network, and the behavioral output (Fig 6).

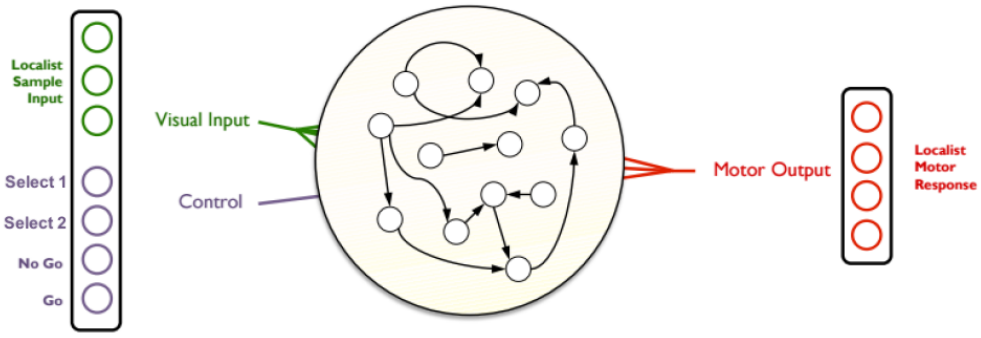


Figure 3: Architecture of the RNN.

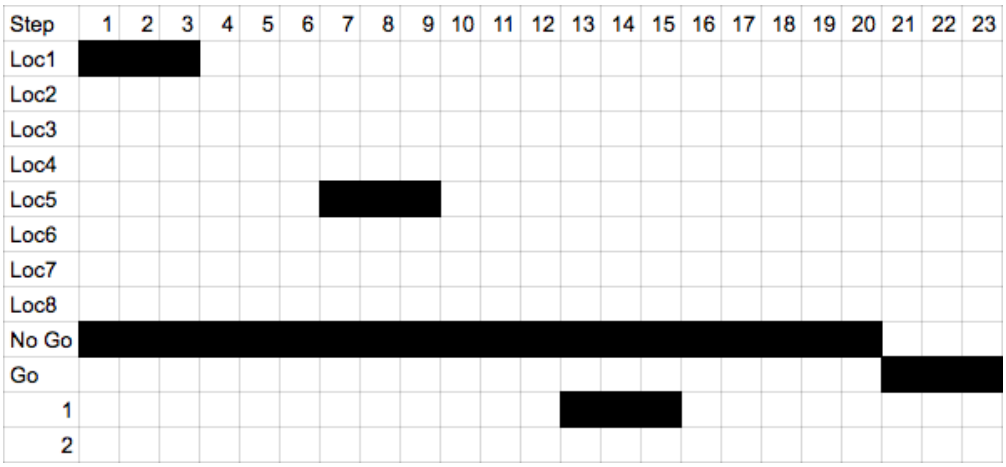


Figure 4: The progression of inputs in an example selection task trial with 8 locations. The first 8 units represent the 8 locations while last 4 represent the control parameters. Sample 1 is presented first, then sample 2, then the selection parameter is activated (in this case, choice 1). The "No Go" unit, which represents the fixation point, is active throughout the presentation of stimuli. Finally, the fixation point disappears and the "Go" stimulus is activated, signaling output presentation.

Step	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
dx+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
dx-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dy+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dy-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5: Example output for the inputs in Figure 4. In this case, location 1 lies on the x-axis, so $-dx, +dy$, and $-dy$ are all 0, while $+dx$ is one for the output presentation.

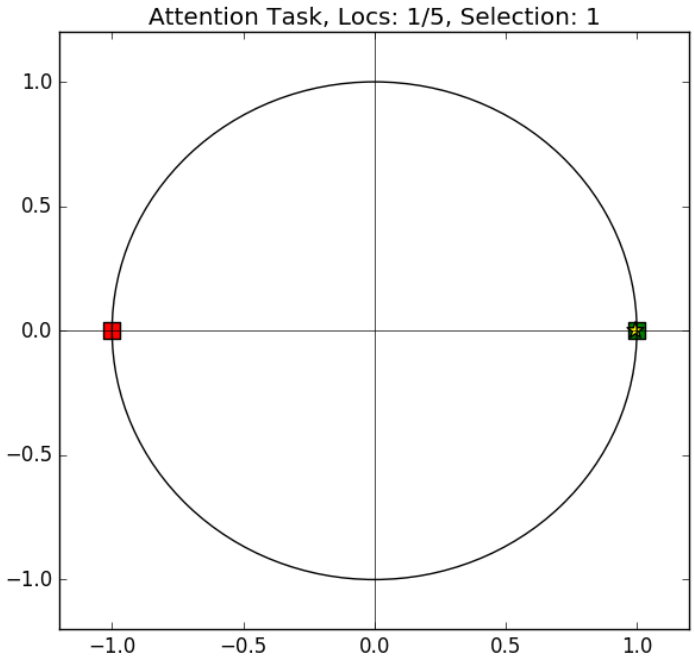


Figure 6: Example behavioral output for the inputs in Figure 4. The green square is the target location, the red square is the unattended location, and the gold star is the response output (in this case, it overlaps the target, indicating a successful trial).

2.2 Dynamics

The network was modeled using nonlinear firing rate units with the following update equations. For the recurrent units,

$$\mathbf{x}_{t+1} = h(\mathbf{W}_r \mathbf{x}_t + \mathbf{W}_i \mathbf{u}_{t+1} + \mathbf{b}_r) \tag{1}$$

Where $\mathbf{x}_t$ is a vector of hidden layer activations at time t, $\mathbf{W}_r$ is an $N_r x N_r$ matrix of recurrent weights between the N_r hidden layer units, $\mathbf{W}_i$ is the $N_i x N_r$ matrix of weights between the N_i input units and the N_r recurrent units, and $\mathbf{b}_r$ is a $1 x N_r$ vector of offset activation biases for the recurrent units. $h(\mathbf{x})$ is the nonlinear firing rate function. Here we use the logistic function, but other options such as the similar $tanh(\mathbf{x})$ function may also be

used.

$$h(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} \tag{2}$$

The output layer units are modeled by a similar equation, with the exclusion of the recurrent term.

$$\mathbf{y}_{t+1} = h(\mathbf{W}_o \mathbf{x}_{t+1} + \mathbf{b}_o) \tag{3}$$

Where $\mathbf{y}_t$ is a vector of output layer firing rates at time t, $\mathbf{W}_o$ is a $4xN_r$ matrix of output weights between the N_r hidden layer units and the 4 output units, and $\mathbf{b}_o$ is a $1x4$ vector of output activation biases.

2.3 Hessian Free Optimization

For many years, the primary algorithm used for training neural networks has been backpropagation of errors. Traditional backpropagation learning algorithms have been very effective when applied to simple feed forward networks with few layers. However, deep learning networks with many layers between input and output are untrainable using simple backpropagation. Because backpropagation works by a first order gradient decent along the error function, repeatedly calculating the first order derivative of the error over many hidden layers causes what is known as the vanishing gradient problem. The error decreases exponentially as it must pass through each new layer. After even just a few layers, the error function becomes arbitrarily small and no learning can take place [10].

While this network is not a deeply multi-layered network, it is highly recurrent, with 19 time steps between the first presentation of input and and the target output presentation. The vanishing gradient problem applies to highly recurrent neural networks as well. One can abstract a recurrent network with many time steps as a simple feed forward network. This

is known as backpropagation through time. A recurrent neural network with n time steps between input and output presentation functions similarly to a deep feed forward network by unfolding each recurrent time step and representing it as another feed forward layer in a deep network. This more clearly shows how input at time step $t = 1$ must propagate through n layers before it reaches the output layer at time step $t = n$.

In order to effectively train a deep or recurrent neural network, James Martens and Ilya Sutskever revised an implementation of Hessian Free Optimization for use on recurrent neural networks. Like most second order optimization algorithms, it uses Newton's Method to compute incremental step sizes to reaching a minimum of some error function. This algorithm uses the Newton CG (iterative conjugate gradient method) to estimate the matrix vector products associated with evaluating Newton's Method, therefore skipping computation of the Hessian Matrix. Because this method is a second order optimization method, it is able to make significantly larger step sizes in areas of low gradient to circumvent the problem of the vanishing gradient [10]. The proliferation of this training algorithm in 2010 has enabled the training of significantly deeper and more complex networks than was previously possible using only backpropagation.

2.4 Training Structure

The first major hurdle was to create a model that didn't just learn the task, but create a model that generalized and performed well on novel inputs. In terms of the bigger picture, working memory is obviously flexible enough to encode novel inputs, thus a model of working memory should be able to as well. Even from a machine learning perspective, a model that incapable of performing well on a novel testing set is a poor model of the data.

Theoretically, a model could be generalizable on any feature of the data. For example, in the case of a delayed saccade task, we could test for with new saccade locations, unfamiliar delay period lengths, or new combinations of familiar features. Given that our

model was trained on fixed delay periods and discrete input directions, we created novel inputs by mixing unseen combinations of known features. For a complete description of the procedure for creating training and testing sets, see Appendix 1.

Given the robustness of Hessian Free Optimizaiton, the network was capable of almost completely eliminating error on the training set in just about every case. However, when given new combinations of features, we were consistently failing to see generalization. The previous literature follows a strong consensus that excessively large hidden layers lead to overfitting and poor generalizability [4], similarly to how excessive regressors in a linear regression leads to overfitting. Thus, we trained a large number of networks with varying hyper parameters (training set completeness, hidden layer size, and number of locations) and found that decreasing the hidden layer size was an the easiest and most consistent solution to create generalization. Below is an table showing a subset of the trained networks, sorted by their errors on the testing set.

Attention Networks	Selection Networks	Combined Networks
Test Error: 5.8e-5, $N_r = 25$	Test Error: 8.7e-5, $N_r = 25$	Test Error: 1.9e-4, $N_r = 25$
Test Error: 2.9e-4, $N_r = 36$	Test Error: 9.3e-4, $N_r = 36$	Test Error: 3.1e-4, $N_r = 49$
Test Error: 0.35, $N_r = 64$	Test Error: 0.42, $N_r = 64$	Test Error: 0.66, $N_r = 64$
Test Error: 0.97, $N_r = 144$	Test Error: 0.96, $N_r = 144$	Test Error: 1.59, $N_r = 144$

Table 1: Testing Error rates on a small subset of networks, showing consistently poor performance as hidden layer size increases. There were also networks of smaller size (¡ 16 hidden units) but these tended to be unable to learn the training set do to limited capacity.

The error value is the same error function minimized during training. We used a simple mean squared error function. That is, for any given set of trials T, the total error is:

$$error(T) = (\sum_{i=1}^{|T|} \sum_{j=1}^{4} (\mathbf{t_{ij}} - \mathbf{x_{ij}})^2)/(2|T|) \tag{4}$$

Where $\mathbf{t_{ij}}$ is the target response from output unit j during trial i, and $\mathbf{x_{ij}}$ is the output activation of unit j during trial i produced by the network, 4 is the number of output

units (4 directions) and $|T|$ is the number of trials in set T. An important observation to note here is that the error is computed across the entire time course of a trial, not just during the output presentation.

3 Methods

Here, I briefly discuss some methods used in our network analysis. Many will be discussed in greater detail in the Results section.

3.1 Linear Regressions

One of the first methods we used to explore networks dynamics was various forms of linear regression. We used linear regressions to predict the activations of each unit in terms of various task parameters and featural information. In general, linear regressions take the form:

$$x_{ij} = \beta_0 v_0 + \beta_1 v_1 + ...\beta_n v_n \tag{5}$$

Where β_i corresponds to the regression coefficient of variable i, v_i. The major difference between different forms of linear regression is the error function used to fit the data. We tried various forms of linear regression, including Ordinary Least Squares, Ridge Regression, Elastic Net, Lasso Regression, as well as cross validated versions of Ridge and Lasso. Additionally, we attempted to apply these linear regression coefficients to a simplified version of the dimensionality reduction and orthogonalization method used by Mante and Sussillo [9].

3.2 Principle Component Analysis

Principle component analysis is a dimensionality reduction procedure that constructs an orthonormal basis from a higher dimensional space, choosing first the dimension that

explains the highest amount of variance. This is especially useful for finding the principle activation space of various task parameters to visualize progressions through activation space.

3.3 t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction algorithm used primarily to visualize high dimensional data in a 2D or 3D space. The key difference between t-SNE and other dimensionality reduction algorithms is that it attempts to find a subspace that maximizes the distance between dissimilar points while clustering similar points. This proves very useful for visualizing neural network data, as high dimensional representations in neural space can be compared neatly in a low dimensional scatter plot. Points in the population activation space that are similar, presumably, are representing similar information. Thus, one can easily visualize the featural aspects of the data on which the neural population divides the activation space.

3.4 Representational Similarity Analysis

Representational Similarity Analysis (RSA) is a method of comparing objects by comparing their similarity (or dissimilarity) matrices rather than comparing them directly. Presented by Kriegeskorte et al as a means of comparing highly dissimilar measurements of overlapping information, it is essentially a method of characterizing the information carried by a given representation in a brain or model. This characterization, through representational similarity (or dissimilarity) matrices (RSM/RDM), can then be compared to other models or regions, completely independent of their original structure. This method has an advantage over first-order correlation techniques in that it finds correlations between similarities rather than absolute representations, matching patterns of representation rather absolute values [7].

In the case of this network, we are not comparing representations across separate models or brain regions. Rather, we are comparing model representations to purely featural representations to see if certain features exist within the activation space of the network.

The process is quite simple. In order to compare two objects, you calculate a trial by trial similarity matrix using any form of similarity measure that fits your data. For example, to compare the activation pattern of a single unit in our model to the presence of a location in a trial, we create two RSMs/RDMs. For the model, we need to find a similarity metric to compare a neural activation to another neural activation. Kriegeskorte et al suggest the simple Pearson correlation coefficient. Thus, the RSM, $\mathbf{M}$, for our vector of neural activations, $\mathbf{n}$ will be calculated as such:

$$\mathbf{M}_{ij} = \rho(\mathbf{n}_i, \mathbf{n}_j), i, j \in [1, |T|] \tag{6}$$

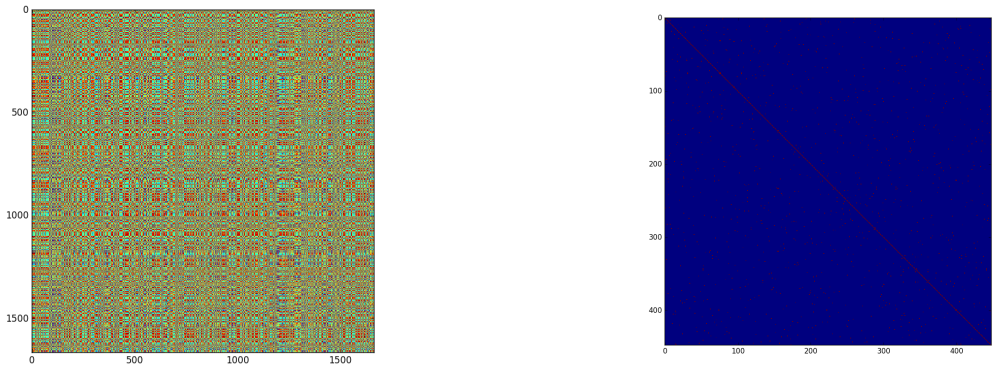
Where ρ is the Pearson correlation coefficient function and $|T|$ is the number of trials. The creativity in this algorithm comes in designing a featural similarity matrix, $\mathbf{N}$. One simple way to do this would be a simple binary comparison. $\mathbf{N}_{ij} = 1$ if and only if both trials contain your feature of interest. Otherwise, $\mathbf{N}_{ij} = 0$. Both $\mathbf{N}$ and $\mathbf{M}$ are diagonal matrices, symmetrical matrices. Finally, to get a single similarity value for your feature, you can again compute the Pearson correlation coefficient between the two matrcies.

$$sim(feature, neuron) = \rho(\mathbf{M}, \mathbf{N}) \tag{7}$$

To see the significance of this coefficient, you can easily generate a random distribution of coefficients by shuffling the trials on one of your objects, then recomputing the RSM and the Pearson correlation. By shuffling the representations and trial pairings, you assure that there is no relationship between the similarity of two trials and the actual similarity in representation. Doing this 1000 times generates a satisfactory distribution against which

you can compare the significance of your similarity value.

Below is are two example similarity matrices. The left matrix is a simple correlation matrix of trial by trial network activation correlation coefficients. The right matrix is a similarity matrix for the existence of a location within a trial. If two trails both have the location, their similarity is 1. Otherwise, it is 0.



(a) Network Similarity Matrix

(b) Feature Similarity Matrix

Figure 7: The left matrix is a simple correlation matrix of trial by trial network activation correlation coefficients. The right matrix is a similarity matrix for the existence of a location within a trial.

3.5 Code and Framework

The current model was developed entirely in python. Some initial code was provided by post doctorate fellow Daniel Rasmussen for a general Hessian Free Optimization framework of recurrent neural networks [14]. All analyses were done using open source machine learning python libraries including sklearn, numpy, and scipy. All figures were generated using matplotlib.

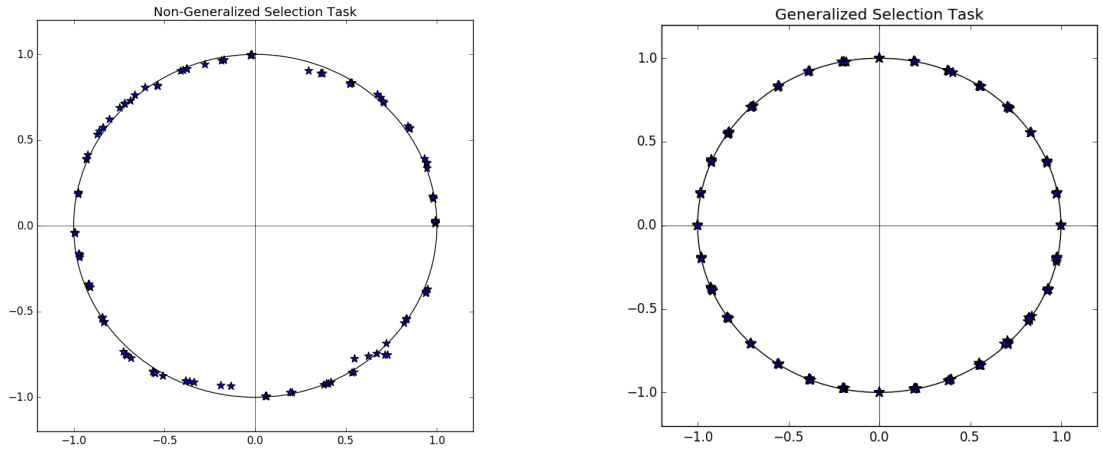
4 Results

4.1 Generalization of Model

Given that we have observed relatively consistent generalizability with a high number of locations and a small hidden layer, numerous questions arise that merit exploration. What is it about this "sweet spot" in model size to data ratio that leads to a more appropriately generalized model? What can we observe in generalized networks that is not present in the failed networks?

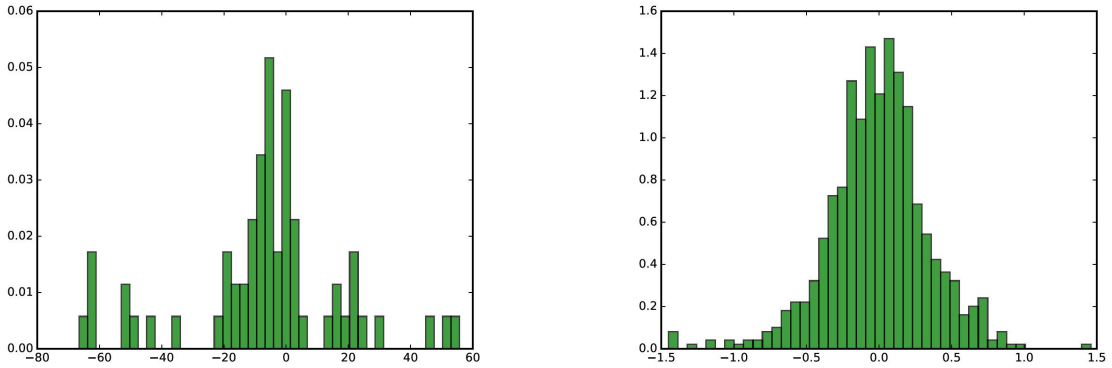
First, in Figure 8, we look at the error plots produced from non-generalized and generalized networks. 8.a shows the typical responses from a non-generalized network while 8.b shows the responses from a generalized network. Note here that the types of errors produced by an overfitted model are not the same types of errors seen in behavioral tasks. Rather than error distributions around the target location, most errors are catastrophic failures. That is, they are not located near either the target or the false cue, but rather some intermediate location. This was observed early on in training.

Typically, a novel combination of target and false cue would result in a saccade in between the two locations rather. This can be observed in Figure 8, which is a histogram of error angles of the saccades. Note, the axis limits are much smaller on the generalized model. In the generalized case, there is a nice, neat distribution of errors around 0, while in the non-generalized case, the error distributions are disjoint. In addition, it is interesting to note in Figure 8 that the network rarely leaves the unit circle. When errors are made, they almost always appear in the continuous circular space, though not at the right location.



(a) Non-generalized (b) Generalized

Figure 8: Generalized vs Non-generalized response patterns to 32 locations. a) shows the typical responses from a non-generalized network while b) shows the responses from a generalized network.



(a) Non-generalized (b) Generalized

Figure 9: Generalized vs Non-generalized theta error distributions. The x-axis is difference in degrees between the target and response. b) shows a neat Gaussian-like distribution around 0 for the generalized network while a) shows disjoint responses in the non generalized network.

To explore why some networks are generalizing while other are not, we used t-SNE to project the activations spaces at each time step down into two dimensions that can be easily visualized. Each time step was bucketed with all other identical steps from all other trials.

Thus, each subplot clusters trial activations by similarity of representation across features present at that time step. The usefulness of t-SNE here is that it will cluster trials with similar activation patterns, showing what features the network is dividing up the activation space on, and on which time steps. In the projections below, the time course of the trials were changed slightly. The pauses between stimuli were removed to make the visualization more clear.

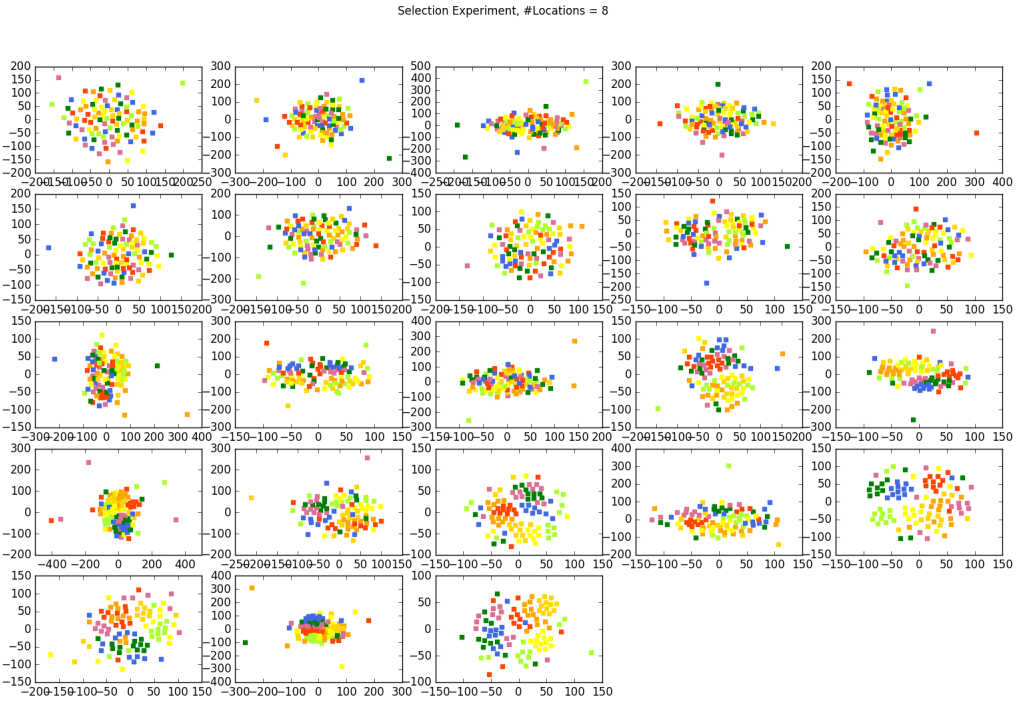


Figure 10: Projection of t-SNE dimensionality reduction through a non-generalized selection task. The first row corresponds to the presentation of sample 1 location. The second row corresponds to the presentation of sample 2 location. The next 3 boxes represent the selection cue presentation. The last 5 boxes are the the trial activations during the response output.

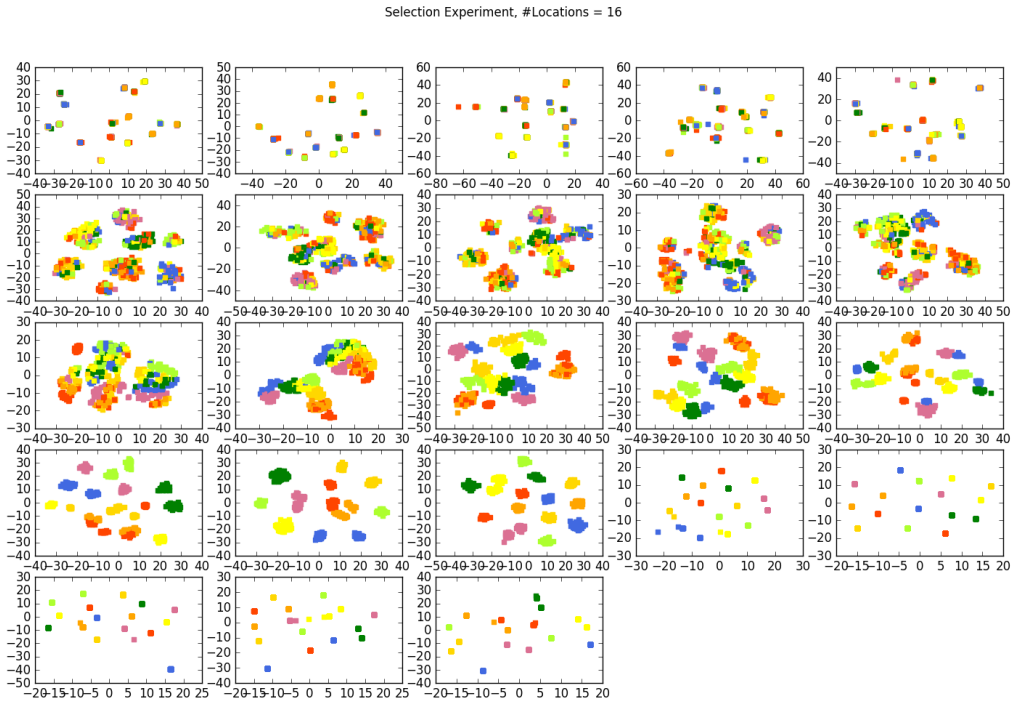


Figure 11: Projection of t-SNE dimensionality reduction through a generalized selection task. The first row corresponds to the presentation of sample 1 location. The second row corresponds to the presentation of sample 2 location. The next 3 boxes represent the selection cue presentation. The last 5 boxes are the the trial activations during the response output.

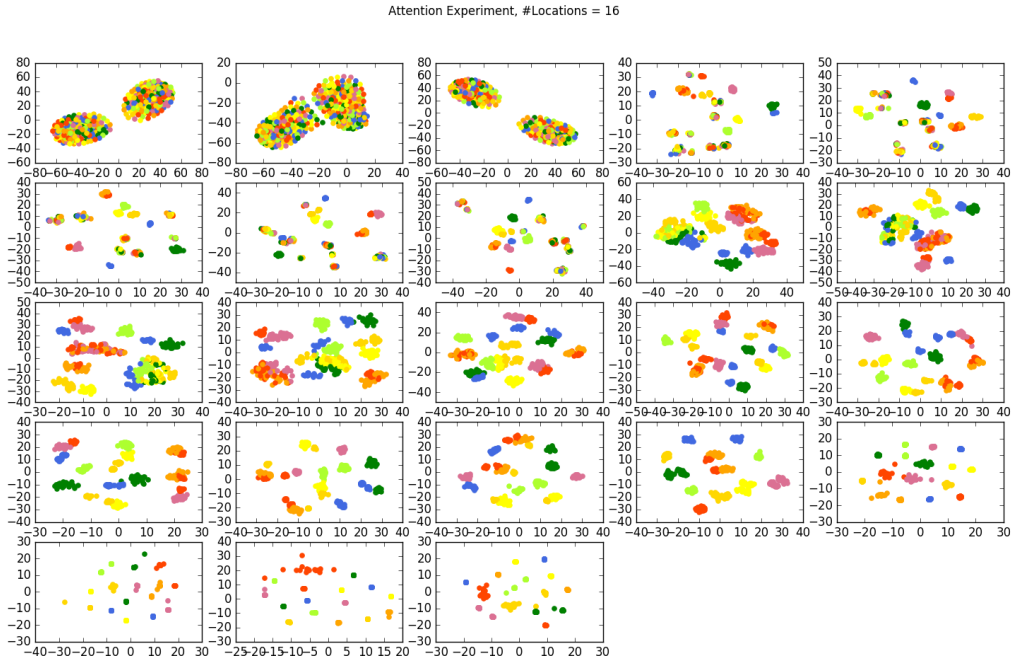


Figure 12: Projection of t-SNE dimensionality reduction through a generalized attention task. The first 3 boxes correspond to the presentation of the selection parameter. The next 5 boxes correspond to the presentation of sample 1 location. The next 5 correspond to the presentation of sample 2 location. The last 5 boxes are the the trial activations during the response output.

The most striking observation here is that in the non-generalized selection task, t-SNE was unable to cluster activations in any meaningful way. The only pattern observable in the non-generalized progression is that similar locations are being represented close to each other. In all progressions, the color of the point represents the eventual answer location presented by the network. Colors closer on the color spectrum represent points that are closer in the output space. The only similarity between points in the Figure 10 progression is their eventual output location. At the very least, this implies that the network is representing some form of continual output space for the 32 locations. However, beyond this, trial activations at every time step appear to be equally similar/dissimilar to all other activations.

Contrast this with the projections in Figures 11 and 12 and it we begin to see a

major difference between generalized and non-generalized networks. At each time step, the 2D projections in the generalized networks divide and cluster the data along the primary feature variables present at that time point. For example, obviously in the attention task, the first 3 time steps immediately divide activation space by the selection parameter. During the selection presentation in the selection task, we see a similar dynamic. In addition, the two subspaces divided by the selection parameter are further neatly sorted by their location (color) similarity.

Perhaps unsurprisingly, the reason generalized networks are able to "solve" novel combinations of features is that, rather than learning the feature to output mappings directly, they learn the underlying structure of the input data. The uniform similarity/dissimilarity between activations in the non-generalized case implies that the solution is little more than a "look up" table, mapping inputs to outputs directly while ignoring the relationships between inputs themselves. This is the same explanation of overfitting found in other machine learning algorithms.

4.2 Selectivity of Units

A simple question to ask is, are representations distributed between neurons, or do subsets of neurons disproportionately represent certain features of the task? One way to measure selectivity of individual neurons is through linear regressions, as described above. The first linear regression design we took broke up the task variables into binary categorical variables, 1 if a feature was present, or 0 otherwise. We then created regression coefficients for a) each of the N locations, b) one for the selection cue, c) one for each location-sample conjunction, d) one for each location-location conjunction, and one for each selection-location conjunction. For example, with 8 locations, this results in $8(locs) + 1(selection) + 2 * 8(sample - loc) + \binom{8}{2}(loc - loc) + 2 * 8(loc - cue) = 59$ regressor variables. Hypothetically, the strength of regression coefficients for each of these binary variables would represent the

relative selectivity of a unit to a given task variable.

We originally had little very little success with basic forms of linear regression, such as OLS and Ridge. In most cases, the regression coefficients came out arbitrarily large or highly inconsistent across different iterations of a model, or even highly inconsistent within combinations of training sets from the same model. This is indicative of high multicollinearity of regressor variables. Given the relationship between locations, samples, and output locations, its unsurprising that multicollinearity was a problem for our data set. To mitigate this, we explored other linear regression models that deal with multicollinearity. For example, the Ridge regression introduces a penalty, or normilzation term in the error function in order to reduce high coefficient values. It attempts to optimize:

$$min_w ||\mathbf{x}w - y||_2^2 + \alpha||w||_2^2 \tag{8}$$

The important addition to ridge is the penalization of the weights by the addition of the L2 norm of the weight vector multiplied by some penalty parameter α. While this is sufficient for reducing unnecessarily large values, the L2 norm (or just euclidean distance) will attempt to spread values as much as possible to avoid the squared penalty on larger coefficients. Given that we are looking for high selectivity in a low number of neurons, the L2 norm isn't ideal. The other regression, Lasso, has a similar optimization function:

$$min_w \frac{1}{n_{samples}} ||\mathbf{x}w - y||_2^2 + \alpha||w||_1 \tag{9}$$

Now, instead of the L2 norm as a penalty term, Lasso uses the L1 norm, or Manhattan distance. This avoids biasing towards distributed coefficients. With the addition of cross validation, LassoCV gave the most consistent regression coefficients. The resulting coefficient vectors for each unit, across most trials, showed that selectivity remained highly distributed across the network. Given that the successfully trained generalized models have so few units

in the hidden layer, it isn't feasible to distribute representations into smaller populations of units. Rather, population level dynamics appears to be the main means to solving the generalized task. Similar results were verified in an RSA analysis. That is, every neuron was highly selective for all locations and the selection cue, and with relatively equal correlation coefficients.

4.3 Dimensionality of Solution Space

Another question we wanted to ask was, what is the dimensionality of the representation? By this, we mean, does the population represent individual locations similarly to conjunctions of locations and other features? Or are the conjunctions of parameters themselves being encoded, with highly disjoint representations for each conjunction? What about on a neural level? To answer these questions, we can look back at the results from the LassoCV regression. Consistently, the regression coefficients for every conjunctive or interactive term were fit to zero, suggesting neurons are selective only for features, while conjunctions of features are created by combining multiple representations.

4.4 Attractor Spaces

The first step in constructing a narrative on how the network might be solving this task was to simply put the network into different activation states that it encounters during the task and then allow it to relax by updating activations until the network a) slows to below some threshold, b) begins bouncing between activation states, or c) exceeds some iteration cut off. Very early on, many time steps during both tasks appeared to be relaxing into an oscillatory behavior. To get a better idea of what was happeing in the network, we began looking for an appropriate subspace to plot the projection through time of the relaxation dynamics.

First, we attempted to implement a simplified version of Mante and Sussillo's dimensionality reduction through creating a regression subspace. We fit a new linear regression on a small set of continuous, independent task variables that would become our primary dimensions. The new regression was of the form:

$$x_{ist} = \beta_{is0} cos(\theta(ans_t)) + \beta_{is1} sin(\theta(ans_t)) + \beta_{is2} cos(\theta(sample1_t)) + \beta_{is3} sin(\theta(sample1_t)) + \beta_{is4} cos(\theta(sample2_t)) + \beta_{is5} sin(\theta(sample2_t)) + \beta_{is6} select_t + \beta_{is7} \quad (10)$$

Where x_{ist} is the firing rate of unit i at step s during trial t, ans_t is the answer location of trial t, $sample1_t$ is the sample 1 location of trial t, $sample2_t$ is the sample 2 location of trial t, and $select_t$ is the selection cue of trial t (either 1 or -1). In their paper, Mante and Sussillo perform a variation of least squares regression to get the vector of beta weights, $\beta_{\mathbf{vs}}$, of length number of units. As seen earlier, this linear regression did not perform well in our case. Instead, we attempted to use the coefficients found from our LassoCV regression to form the vector of beta weights. Because they were working with noisy population neurons, Mante and Sussillo also applied a targeted dimensionality reduction through the first 12 principle components of the population z-scored responses. Given our model had no noise and only $25 - 36$ hidden units, we skipped this step.

Next, for each variable, we attempted to find the time step s that produced the most significant set of beta weights for the population, thus "best" representing that variable. We used the max L2 norm of the beta weights.

$$\beta_v^{max} = \{\beta_{vs} | s = argmax_s ||\beta_{vs}||_2^2\} \quad (11)$$

Naturally, the time steps with the greatest norm for any given task parameter was the time step in which the task parameter was presented. The final step was orthogonalizing

the regression subspace using **QR** decomposition.

$$\mathbf{B}^{max} = \mathbf{QR} = [\mathbf{B}_0^{max}, \mathbf{B}_1^{max}, \mathbf{B}_2^{max}, \mathbf{B}_3^{max}, \mathbf{B}_4^{max}, \mathbf{B}_5^{max}, \mathbf{B}_6^{max}] \tag{12}$$

We attempted to project activations into this space with little success. We also attempted to use the non-orthogonalzed subspace, β_v as well, with equally poor results. There are few reasons why this space was likely unhelpful in representing our activations. 1) Mante and Sussillo were working with unrelated task variables (color and motion) and orthogonalizing their regression weights didn't effect the representations. In our task, it is very possible that the features are simply not orthogonal or that there is high multicollinearity between representations of, say sample 1 and sample 2. As we will soon see with future analyses, this certainly seems to be the case. 2) The task in Mante and Sussillo's paper (information integration task) was a relatively stable task from start to finish. This led to a highly stable regression subspace as a basis for the data. In our model however, the task is highly dynamic. At each phase, it is very likely that the network is completely shifting its representation space. Performing an analysis similar to Mante and Sussillo, neither the orthogonal regression space or the straight beta vector stayed stable through different phases of the trial.

We instead moved to using simple principle components as dimensions to reduce the activation space. While observing relaxation patterns of in the selection model, it became clear that the selection cue was not choosing one representation in working memory and collapsing the other, as hypothesized. Instead, the selection cue appeared to be acting as a timing mechanism that stopped an oscillation over some representation space. For example, Figure 13 shows a typical selection trial, only the selection cue was shifted forward one time step.

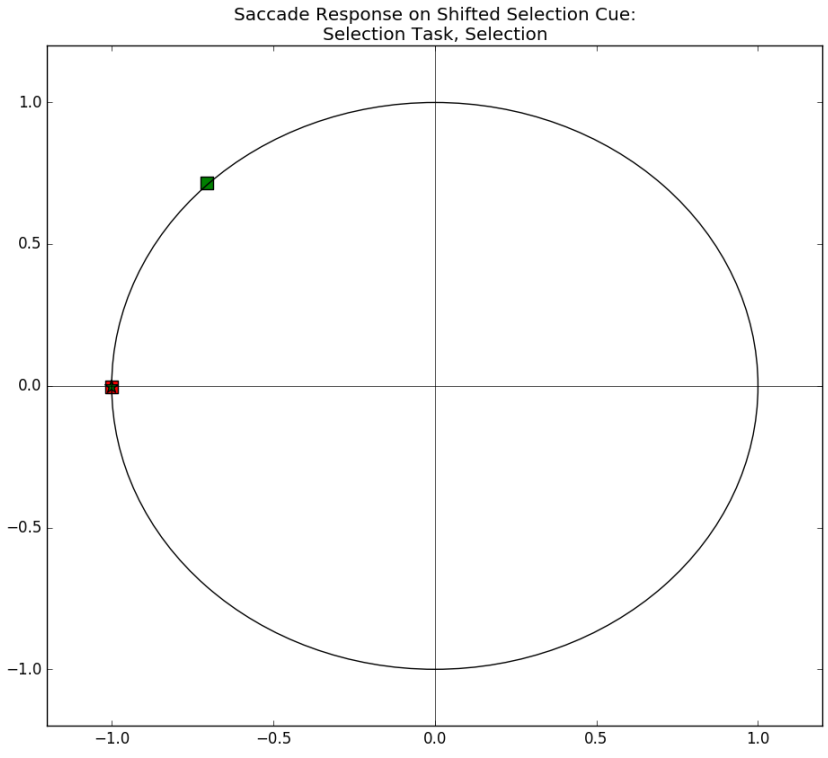


Figure 13: A generalized selection network performing a typical saccade task. However, the presentation of the selection cue was held for an extra time step before switching off. The green location is the target location, the red location is the false location, and the green star is the actual saccade output. Note that the selection value was not changed (it still asked for the same location), only changed in timing by one step. The resulting saccade is to the incorrect location.

Given this evidence, we wished to explore the possibility that the network was oscillating between the two presentation locations (sample 1 location and sample 2 location). To test this, we projected the activations into a 2D space and relaxed the network throughout the time course of the trial. The two principle component dimensions represent the sample location 1 and the sample 2 location. These were fit by bucketing all hidden layer activations during the presentation of the respective sample only, and calculating the principle component. The results can be seen in Figure 14.

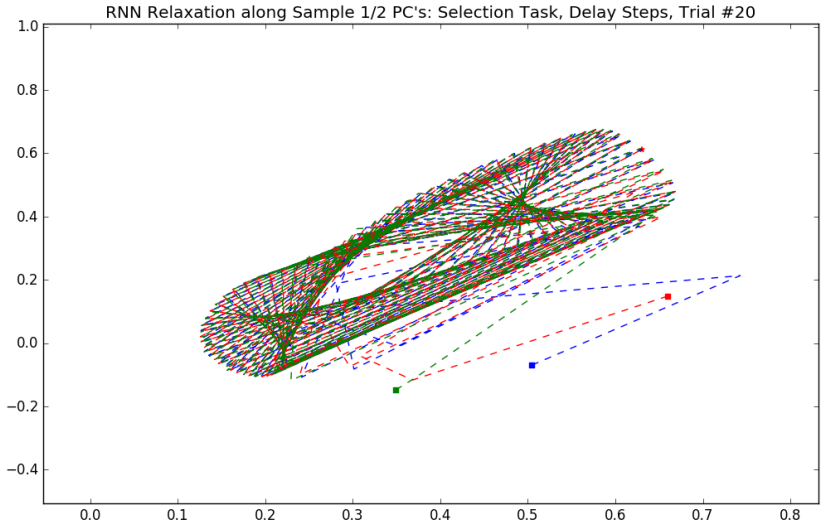


Figure 14: The relaxation of the recurrent layer during the delay period. Each point (blue, red, green) represents a time step during the delay period from which the relaxation started. Notice the *seemingly* oscillatory nature of the trajectories along the 2 principle components.

Initial results showed an oscillatory like behavior of the sample as well as a slow drift down into a higher dimensional attractor space. After drifting down into the attractor space, the activations continue to bounce along the diagonal indefinitely. Obviously, this immediately raises many questions. First, does shifting the selection cue lead to the same projection, but slightly out of phase, as our new hypothesis timing would suggest? Indeed, shifting the selection parameter back or forward one step in the trial then relaxing activations in the delay period put the activations exactly one step out of phase with the typical relaxation response in the network as seen in Figure 14, however the activations circled the exact same attractor space.

If the selection cue is responsible for timing the movement of the activation, can we see some timing dynamics from the selection step? We performed a similar analysis, but relaxed the network from the selection phase instead of the delay period, as seen in Figure

15.

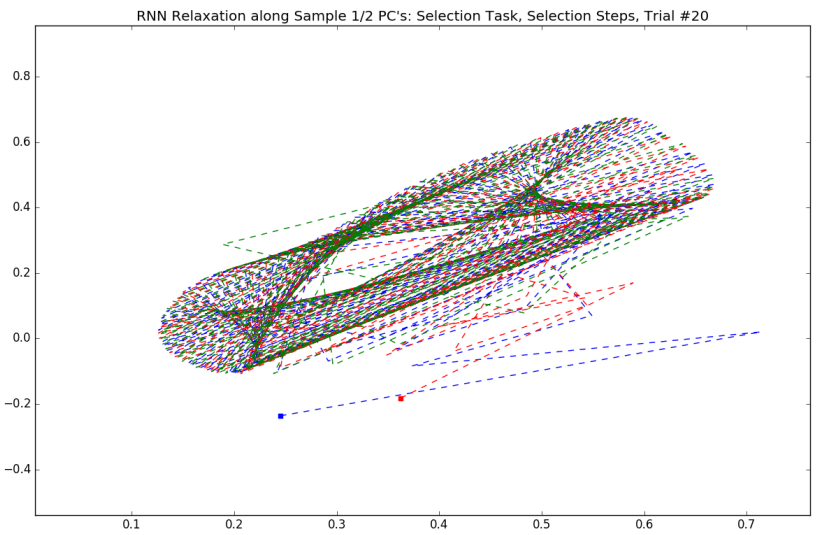


Figure 15: The relaxation of the recurrent layer during the selection period. Each point (blue, red, green) represents a time step during the selection period from which the relaxation started.

Unfortunately, this new projection did not tell us much about the dynamics of the selection cue. To try to see more of the selection dynamics, we wanted to look for a 3D dimension that might explain more of the attractor space. In Figure 16, we chose a 3rd PC to represents the selection cue parameter, and again plotted the relaxtion projections from each selection step.

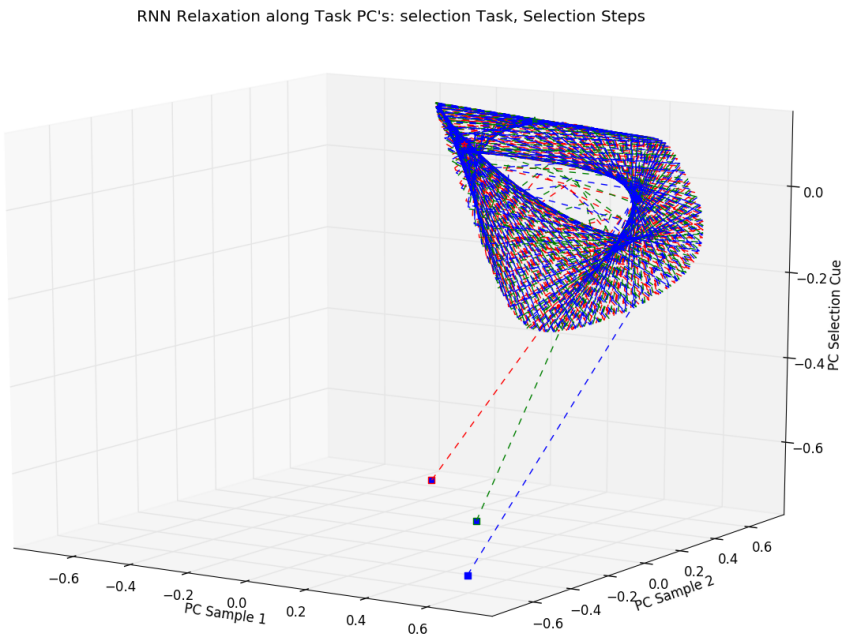


Figure 16: The relaxation of the recurrent layer during the selection period. Each point (blue, red, green) represents a time step during the selection period from which the relaxation started.

In this new space, we can begin to see more of the attractor dynamics. While the selection cue is active, the activation is held in a relatively unstable position. Once the selection cue "releases" the network, it relaxes towards the attractor space. Once projected in this new 3D space, it became clear that this attractor space was closer to a pointed triangle or circle rather than a true oscillation. If the selection cue simply holds the network and then releases it to begin a circling the attractor space, what differs between the two selection parameters? To answer this, we plotted, as above, the same relaxation in 3D space during the selection period, but we also swapped the selection cue and redid the trial again with the flipped selection. The two projections are plotted in the same space in Figure 17

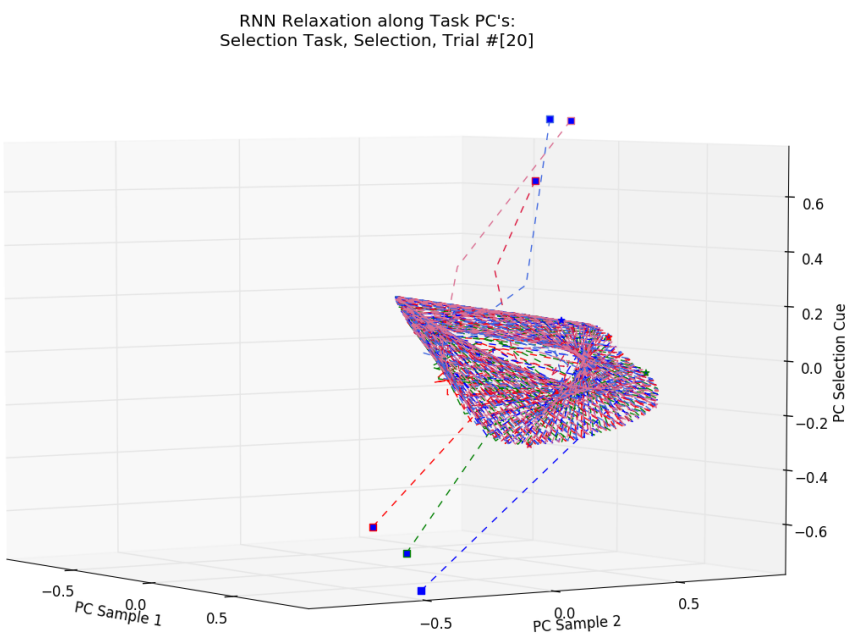


Figure 17: The relaxation of the recurrent layer during the selection period. The top three points represent relaxation from one selection cue while the bottom three points represent relaxation from the opposite selection cue.

While the two selection cues clusters begin on opposite sides of the attractor space, they both quickly drift onto the attractor space, albeit in different places. Thus, it appears that the selection cue decides where the activation enters this attractor space, while the position on this attractor space determines output location. We were also curious if this attractor space was consistent for all trials. The answer is a absolutely yes. We drifted the model from various time steps during every trial, and we found that if the activations are drifted anytime between the presentation of the second location and the cue for a motor output, they invariably fall very quickly onto this attractor space and begin circling it. To see what this attractor space corresponds to motor wise, we drifted the activations into the attractor space and at each point, forwarded the activation to the output layer and plotted the motor response. The results are in Figure 18

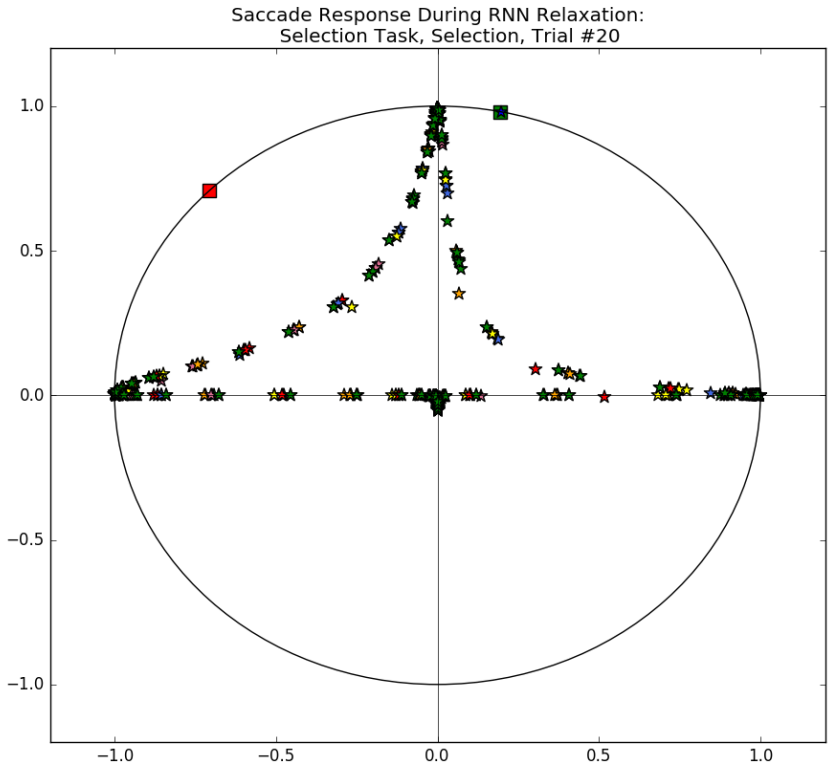


Figure 18: The output response corresponding to the drifting hidden activations throughout the circular attractor space.

Finally uncovering the dynamics of the selection model explains why we were having so much trouble with the orthogonal regression subspace earlier. There appears to be a single, underlying representation of locations, and the sample cues only determine where to enter the space. In that sense, the sample 1 and sample 2 locations are not at all orthogonal, and in fact project down onto the same space.

Do other selection tasks create similar solutions? Yes, this seems to be a consistent solution across multiple generalized selection networks. Figure 19 shows another attractor space, created using the same plotting mechanism, but with a different network.

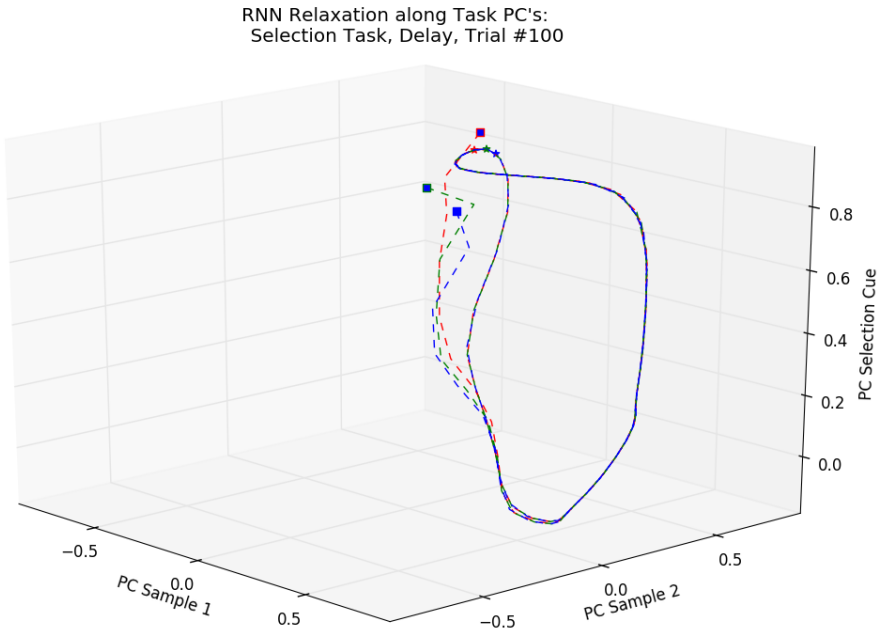


Figure 19: The relaxation of the recurrent layer during the delay period.

Additionally, we found a different attractor space before the presentation of the second location. This attractor space is identical across all locations as well. It appears that the network settles into a location representation from the first location, then drifts towards the combined attractor space from this old location, once released by the cue selection. Below two separate input locations that approach the space.

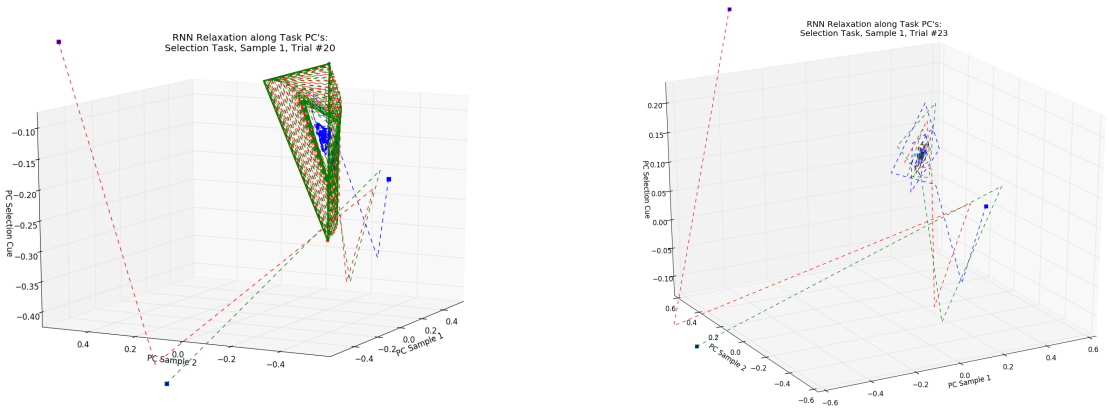


Figure 20: Two attractor spaces for different sample 1 locations.

Note, separate attractor spaces for sample 1 vs sample 2 locations would contradict my earlier explanation of the ineffectiveness of the regression subspace. If sample 1 locations and sample 2 locations have separate attractor spaces, there should be a regression subspace to capture those differences. With future exploration, perhaps letting the network drift on the sample 1 cues, then running a linear regression on the relaxed activations would be more successful.

4.5 Dynamics of Attention Task

The dynamics of the attention task seemed to much less complex, though we did not explore it as thoroughly as the selection space. In particular, moving the unattended location or sometimes removing it entirely often does not affect the performance on a task. This confirms earlier observations that the attention task is able to completely ignore unnecessary, unattended stimuli. As further evidence, consider the figure below:

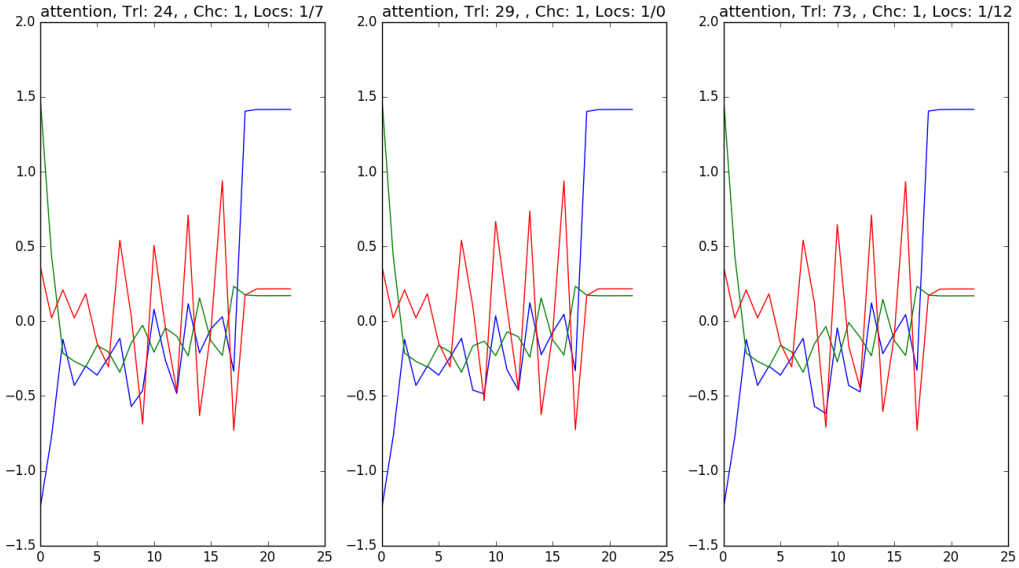


Figure 21: The three principle component activations along 3 separate trials of the attention task.

Notice that despite the three panels being separate trials with separate location combinations, the selection cue and selected location are identical, leading to identical projections of the principle components. This, as well as RSA analysis between trials of the same selection cue to location conjugation, shows that the attention task is effectively ignoring unnecessary stimuli.

However, the attention still has some interesting dynamics. As you can see in the principle component projections, there seems to be a great deal of true oscillatory motion in the attention task, far more so than the selection task. Plotting the activation drifts as before shows interesting results.

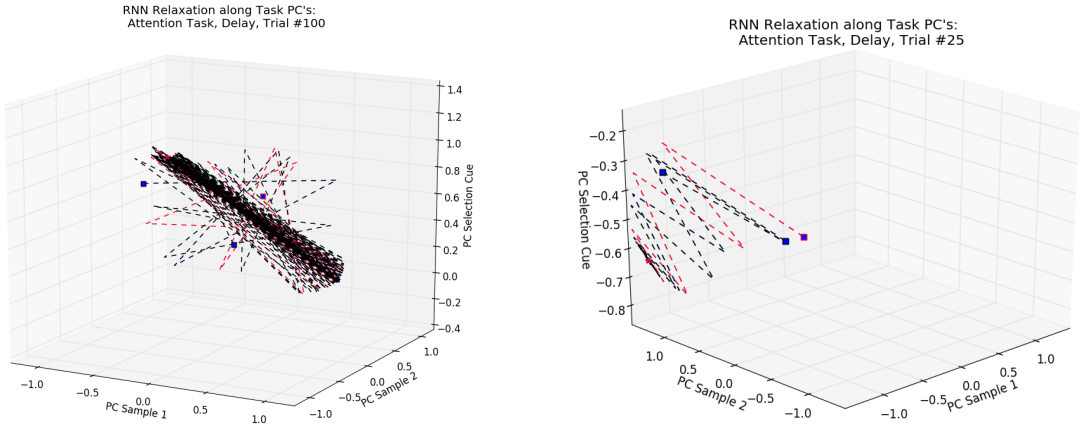


Figure 22: Activation drifts of two attention networks during the delay period. Note that the true oscillatory patterns of these two attention networks. After a few dozen time steps, the first network jumps between two identical points.

4.6 Combined Network Solutions

While the networks admittedly found extremely interesting solutions to the attention and selection problems, the combined networks did not appear to change the solutions, but rather, they learned the two tasks independently and employed a similar strategy to their single counterparts on the appropriate tasks. For example, similar attractor dynamics were observed for selection tasks and attention tasks. Most damming of all separating trials

into attention and selection trials and then performing RSA on the separate activation sets resulted in identical distributions and correlation coefficients to their singly trained counterparts. In other words, the selectivity of the networks to various task parameters remained identical.

5 Discussion

5.1 Conclusions

While the analyses speak for themselves, there are a few closing comments. Given the simplistic architecture of our network and the unconstrained nature of our training, there was significant uncertainty whether the model would be able to find a meaningful solution, or really a solution at all. Not only was a simple recurrent neural layer able to find a meaningful solution, many aspects of the solution, such as stability and attractor spaces, are usually seen in deliberately modeled architectures that predispose stable solutions (such as bump attractor networks) [8]. In particular, the circular attractor space found in the selection networks is surprisingly complex of a solution for such a simple network. Is there some inherent simplicity to such a solution? Is this the most compact representation for the solution space? To some extent, by intentionally finding the smallest hidden layer size capable of performing the task, we are necessarily choosing a compact, low dimensional space. At the same time, as shown by the error tables in the first section, this also seems to be the best solution for generalizing to novel inputs.

What exactly does this mean for working memory? In particular, this demonstrated an interesting divide and conquer strategy, simultaneously representing an underlying space for a data type (spatial locations) and a dynamical system for moving through that space to time a working memory task. Whether or not such a solution would be chosen naturally by the brain is out of the realm of this project. However, this exploration shows the robust

capacity of single layer recurrent networks. Even complex, overlapping stable representations can emerge even in simple, single layer networks. Some aspects of the model which which are less interesting are likely a consequence of the simplicity. For example, in the combined network, tendency to independently learn each task rather than find a novel, combined solution is likely a reflection on the inflexibility of such an unconstrained training structure with a simple single layer network.

5.2 Future Work

There are an endless number of directions future explorations could take, and quite a few more that I thought I could include, but ultimately were left out. The most obvious early inclusions are variability in stimuli presentation and noise.

By variation in stimulation time I mean variation in the delay periods, stimuli presentation, and trial lengths. Most behavior tasks measuring working memory involve uncertainty in timing while this model uses constant time steps. This likely is the reason for the observed oscillatory nature of the selection parameter in the selection task. Moving the selection parameter forward or backward one time step in some trials resulted in a saccade close to the non-selected location. Given fixed delay periods, the network was able to time oscillations between two location representations to match the selection presentation. This rather inflexible solution would fail given uncertainty in the delay period. However, it is interesting to note that studies of attention note an oscillation between attended location items, resulting in oscillating response times.

In the case of noise, most network models include uncertainty terms in the activation update equations (Eq 1.) [8,9,10] or in the input data. Our initial goal was to explore the most simple solutions in unconstrained learning, and thus adding additional dimensions to learning such as uncertainty in the input seemed extraneous. Given that most models attempt to directly model behavioral data, noise is essential given the highly noise in neural

data. Initially, I assumed noise would be necessary to push the network towards a general solution. That is, in the absence of consistent combinations of inputs and outputs, the network would be forced to learn the underlying patterns in the data instead of simple input-output mappings. This perhaps could have aided during learning with over-fitted networks. However, generalization occurred rather easily by changing the size of the input layer, which alleviated the need to explore noise.

Given that fitting behavioral data was not a primary motivation of this study, there are numerous features that were left out that in future studies, could lead to more relevant insights into possible solutions implemented in the brain. For my second Junior Paper last year, I implemented a form of local connectivity in a similar, though much more simple, model. Constrained by physical limitations, cortical networks necessarily follow local connectivity rules. It isn't a stretch to assume that such connectivity heuristics change solutions implemented by cortical networks. These heuristics can be modeled with simple distributions on the existence of connection weights over distance between neurons. Previous studies have found that neural connectivity tends to follow a lognormal distribution over distance in the cortex [15].

6 References

References

[1] Barash, S., R. M. Bracewell , L. Fogassi , J. W. Gnadt , R. A. Andersen, *Journal of Neurophysiology*, Published 1 September 1991 Vol. 66 no. 3, 1095-1108 DOI:

[2] Bisley JW, Mirpour K, Arcizet F, Ong WS. 2011. The role of the lateral intraparietal area in orienting attention and its implications for visual search. The European Journal of Neuroscience 33.1982–1990

[3] Brody CD, Hernandez A, Zainos A, Romo R. Timing and neural encoding of somatosensory parametric working memory in macaque prefrontal cortex. Cereb Cortex 13: 1196 –1207, 2003.

[4] Fyfe, Colin. Articial Neural Networks and Information Theory. 1.2st ed. N.p.: U of Paisley, 2000. Print.

[5] Gnadt, J.W. and Andersen, R.A. Memory related motor planning activity in posterior parietal cortex of macaque. *Experimental Brain Research* 70, 216–220 (1988).

[6] S. Hachreiler, 'The Vanishing Gradient Problem During Leaning Recurrent Neural Nets and Problem Solutions," Inr. J. Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 6, no. 2, pp. 107-1 16. 1998.

[7] Kriegeskorte, Nikolaus, Marieke Mur, and Peter Bandettini. "Representational Similarity Analysis – Connecting the Branches of Systems Neuroscience." Frontiers in Systems Neuroscience 2 (2008): 4. PMC. Web. 29 Apr. 2016.

[8] Ma, Wei Ji et al. "Changing concepts of working memory." (2014).

[9] Mante, et al. Context-dependent computation by recurrent dynamics in prefrontal cortex Nature, 503 (2013), pp. 78–84

[10] Martens, J. (2010). Deep learning via Hessian-free optimization. In Proceedings of the 27th International Conference on Machine Learning.

[11] Miller P, Brody CD, Romo R, Wang XJ. 2003. A recurrent network model of somatosensory parametric working memory in the prefrontal cortex. Cereb Cortex. 13:1208–1218.

[12] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, et al. (2011) Scikit-learn: Machine Learning in Python . Journal of Machine Learning Research

[13] Powell KD, Goldberg ME (2000) Response of neurons in the lateral intraparietal area to a distractor flashed during the delay period of a memory-guided saccade. J Neurophysiology 84. 301–310

[14] Rasmuss. Daniel, Python implementation of Hessian-free optimization, (2015), GitHub repository, https://github.com/drasmuss/hessianfree

[15] Song, S., Sjöström, P. J., Reigl, M., Nelson, S. and Chklovskii, D. B. Highly nonrandom features of synaptic connectivity in local cortical circuits. PLoS Biol. 3, e68 (2005).

[16] Steenrod SC, Phillips MH, Goldberg ME The lateral intraparietal area codes the location of saccade targets and not the dimension of the saccades that will be made to acquire them. 2013; J Neurophysiol 109. 2596–2605

[17] Sussillo, D. Barak, O. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. Neural Comput. 25, 626–649 (2013).

[18] Thier P, Andersen RA (1998) Electrical microstimulation distinguishes distinct saccade-related areas in the posterior parietal cortex. J Neurophysiol 80:1713–1735.

[19] Wimmer, K., Nykamp, D.Q., Constantinidis, C. Compte, A. Bump attractor dynamics in prefrontal cortex explains behavioral precision in spatial working memory. Nat. Neurosci. 17, 431–439 (2014).

Appendix

Training and Testing Set Construction

A given training set has anywhere between 80-100% completeness of task parameter combinations. To balance the training and ensure a parameter is not over-trained, each location, sample, and selection parameter occur together at least once. The variability in comprehensiveness comes from varying the number of possible false cue locations paired with the correctly cued stimuli. The training set is created like so:

p = % percent comprehensiveness

For each location, 'l':

    For each sample, 's':

        For each selection, 'cue':

            choose subset of size p% of the remaining unused locs;

            create trial with s, l, cue, and p% unused locs;

Source Code

For complete access to all code used in this project, see the following public github repository: `https://github.com/cjwawrzonek/Thesis.git`