Dynamic Working Memory in Recurrent Neural Networks

Alexander Atanasov

Research Advisor: John Murray

Physics 472

Spring Term

Abstract

Recurrent neural networks (RNNs) are physically-motivated models of brain circuits that can perform elementary computations. The relationship between the structure of the connectivity and the dynamical output is difficult to determine in all but the simplest of systems. Here, we study the dynamics arising from RNNs that are trained to perform simple discriminatory tasks. In particular, we study the ability of a network to generalize its performance to cognitive tasks whose parameters are outside the space of trials that it was trained on.

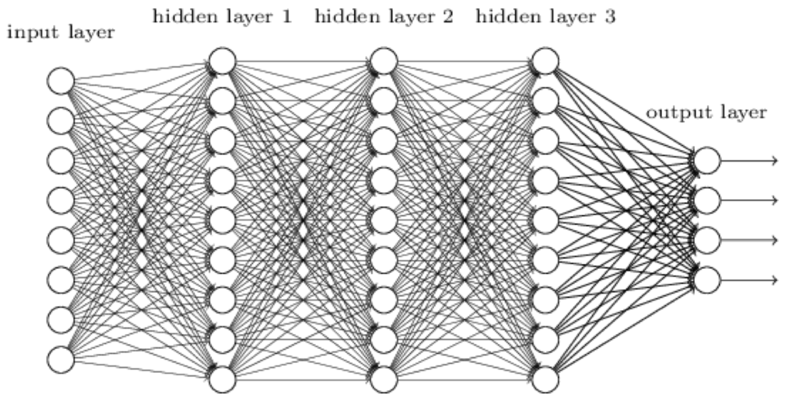
Contents

1 Introduction to Neural Networks

1.1 Feedforward and Recurrent Networks

The study of artificial neural networks has existed already for several decades and finds applications in fields ranging from image detection to language processing. More than this, artificial neural networks (ANNs) provide a looking glass into the ways that neuronal systems in the brain can perform elementary computation.

Within the machine learning community, for many tasks such as language processing, the data is input into an initial layer of neurons, that then connect to a series of hidden layers before returning an output [1], as below.



These are multilayer systems, with several levels of neuron families and no connectivity among the same level, but very high connectivity between successive levels. The inputs of neurons at level i depend only on the inputs for the neurons at level $i-1$, and their firing will only effect neurons at level $i+1$.

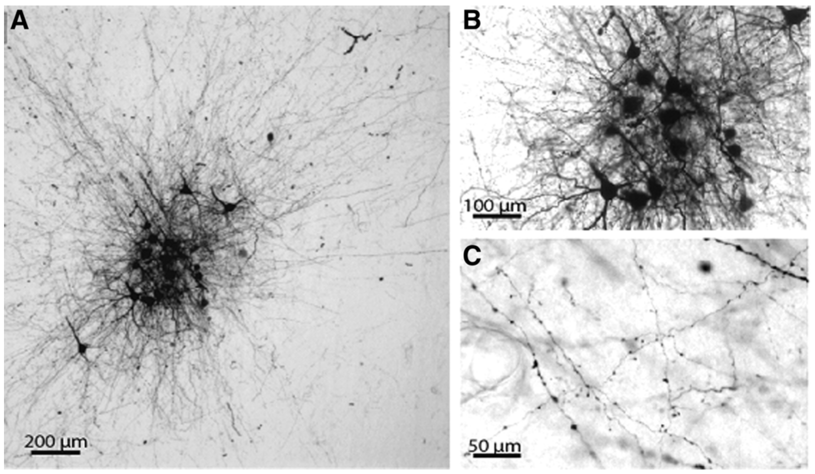
A neuron will have a firing rate r_i depending on whether it is receiving enough current to pass some threshold. This current comes from all the neurons going into it. That is, for each neuron i, a neuron j going into i contributes an amount of current $W_{ij}r_j$, where W_{ij} is the *weight of connectivity* between i and j. This can be negative, in which case we would have that neuron j *inhibits* neuron i. The firing rate of neuron i is then some positive function of the neton's current $r_i = [x_i]_+$ (e.g. a sigmoid or a delayed linear function with some threshold) [2].

There is some output function z_ℓ^* for each output neuron i, that the neural network must be trained to produce. That is, the goal is to minimize $\mathcal{L} = ||z_\ell^* - z_\ell||$. How do we minimize this error? We see how we'd have to adjust the current immediately going in to the output layer by adjusting those weights so that the error function decreases the most rapidly. But the current in each penultimate node is dependent on the layer before *that*, so we must see how to adjust those weights as well in order to minimize the total error. The total error change is then expressed (through the chain rule) in terms of all the weight changes from the final to the initial layer. This is the principle of backpropagation.

Once we have the changes in error expressed in terms of the changes of weights, we find the direction which minimizes the error the fastest. This is the method of steepest gradient descent (which will be explained more deeply in the following sections).

But the human brain does not only consist of networks like the one above [2]. In fact it rarely does. Firstly, there is no dynamic element to these networks. The inputs u_k are fixed, independent of time, and so are the outputs z_ℓ. Biologically, these would be time-varying input/output currents $u_k(t)$ and $z_\ell(t)$

In addition, the networks above are "feedforward", but they are not recurrent. The picture below is a section of a neuronal network from a mouse.



Note that in this picture, the networks do not fit together in separable layers but instead seem to bunch up together in a highly connected set of nodes. In mathematics, such densely connected graphs are called expanders. Recurrent neural networks (RNNs) consist of connectivity among all neurons in the network without separating the neuronal connections into distinct "sectors" as in the first picture.

In certain parts of the brain, there can be a mix of feedforward networks and RNNs, where the network consists of sparsely connected sectors of RNNs, each RNN feeding forward into the next. There is biological evidence for this phenomenon, but we will focus on only RNNs. Similar techniques for training feedforward networks can be applied to RNNs.

A reasonable model for neural networks is given in terms of relating the current in a given neuron x_i to a linear inhomogeneous differential equation. This equation would represent exponential decay of activity were it not for the inhomogeneous terms representing the inputs from the other spiking networks.

$$\tau \frac{dx_i}{dt} = -x_i + \sum_{j=1}^{N} W_{ij}^{\text{rec}} + \sum_{k=1}^{N_{in}} W_{ik}^{\text{in}} u_k + \sqrt{2\tau\sigma^2}\xi_i \tag{1.1}$$

$$r_i = [x_i]_+ \tag{1.2}$$

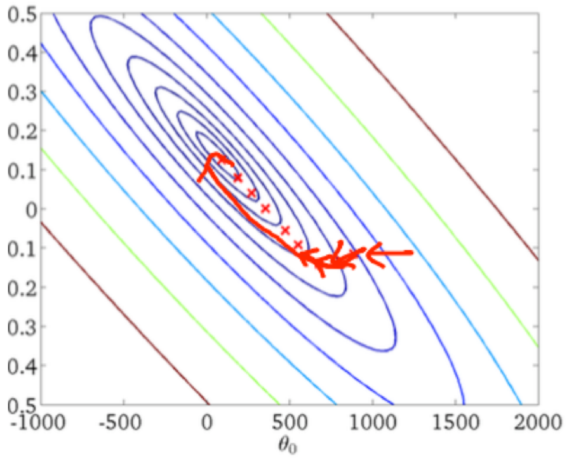
$$z_\ell = \sum_{i=1}^{N} W_{\ell i}^{\text{out}} r_i \tag{1.3}$$

The last term is noise, which is both biologically and computationally significant. The output function $z_\ell(t)$ is obtained from an output weight matrix $W_{\ell i}^{\text{out}}$ acting on the recurrent network.

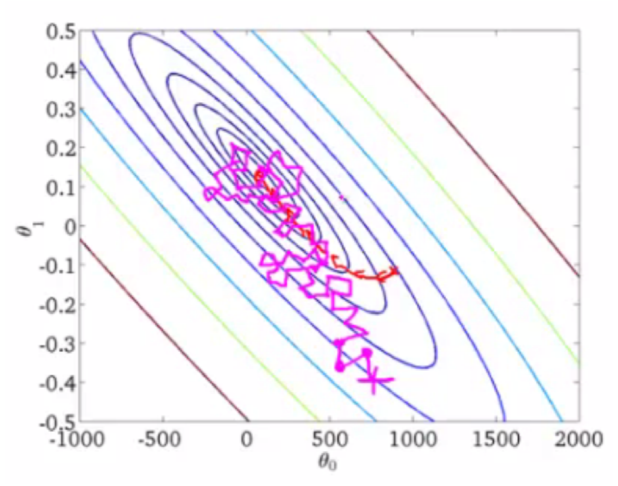
1.2 Training Neural Networks: Backpropagation and SGD

To train a neural network means to appropriately adjust the weight matrices $W_{ij}^{\text{rec}}, W_{ij}^{\text{in}}, W_{ij}^{\text{out}}$ so that for the given input $u_k(t)$ we get an output $z_\ell(t)$ that is as close as possible to the desired $z_\ell^*(t)$. Again since firing rates recursively depend on previous neurons at past time-steps [3], we will have to employ backpropagation as in the feedforward case.

This time it is different: a neuron's own fired current may end up influencing it in the future [4]. This forms feedback loops over each individual neuron. To minimize the error function $\mathcal{L}(W_{ij})$ with respect to the weights, we employ the method of stochastic gradient descent. This method is a variation on the more simple method of gradient descent: given a point in some space, calculate the gradient of the error function (it points orthogonal to the level curves of the error) and move against the gradient (so we go in the direction of greatest decrease).



We continue doing this to converge on a minimum. Note however that this minimum may only be a local minimum and thus not be good enough to minimize the error. How do we rectify this? By adding an element of randomness, thus making the method stochastic.



The idea is that if we are trapped in only a local minimum, with a better one nearby, this randomness can "bump" us out, into the lower minimum.

1.3 The 'Pycog' Package

Many of the methods that were used in this research project to perform the backpropogation and the SGD methods were developed by Francis Song [4] and his colleagues at NYU in the *pycog* package. The code can be accessed publicly at

https://github.com/frsong/pycog.

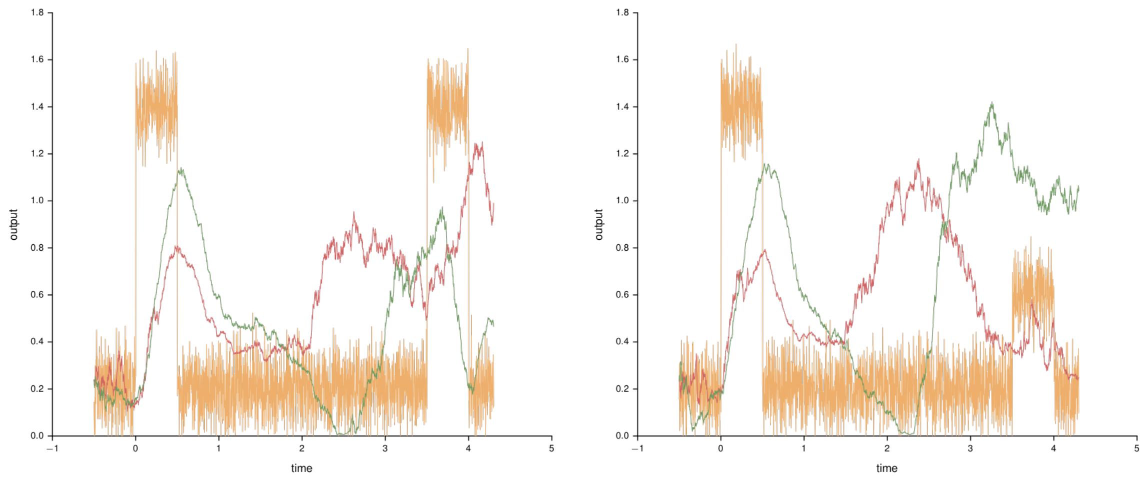
2 Discrimination of Frequencies

The purpose of this project is to train neural networks to be able to hold a dynamic working memory long enough to decide whether two pulses, given at different times, were the same or not. Then, on these trained networks, we wish to be able to perform analysis to determine how the working memory is stored in these dynamical systems.

We can make the pulses be different frequency, or we can make them come from different neurons. These two tasks, while seemingly simple in structure, lead to widely different dynamics and training behaviour.

2.1 The Task of Variant Delay

We have one input neuron feeding in two pulses with a delay period Δ_0 between the two pulses. The pulses may be of differing frequencies.



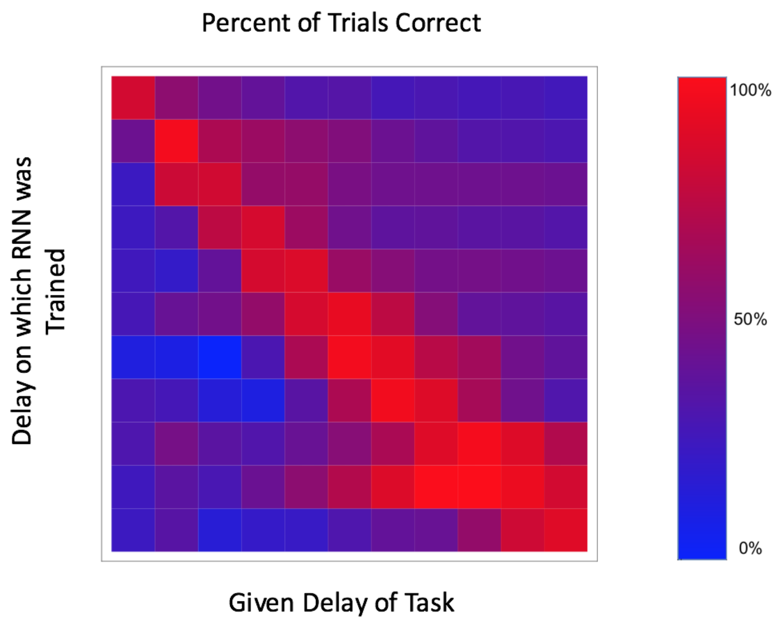
The orange curve represents the stimulus from the input neuron. The red and green neurons represent the two outputs. Note how in the first case the red output neuron ends up high (meaning match) while in the second case the green output neuron does (meaning non-match). This is similar to a previously studied task by Romo et al. in which the network is trained to tell which of the two frequencies is higher. In fact, the training dynamics of this task are radically different from the Romo task. Saying "same" versus "different" is a more complicated process for a network than deciding which is greater.

A series of networks were trained on various delays, ranging from a 0 second delay to a 5 second delay, in increments of .5 seconds. There were 5 networks trained on each delay setting at different random seed. The resultant data for each delay setting is taken as a mean over the networks. In this way, network-specific phenomena caused only by different initializations of weight matrices W_{ij} were smoothed away by averaging.

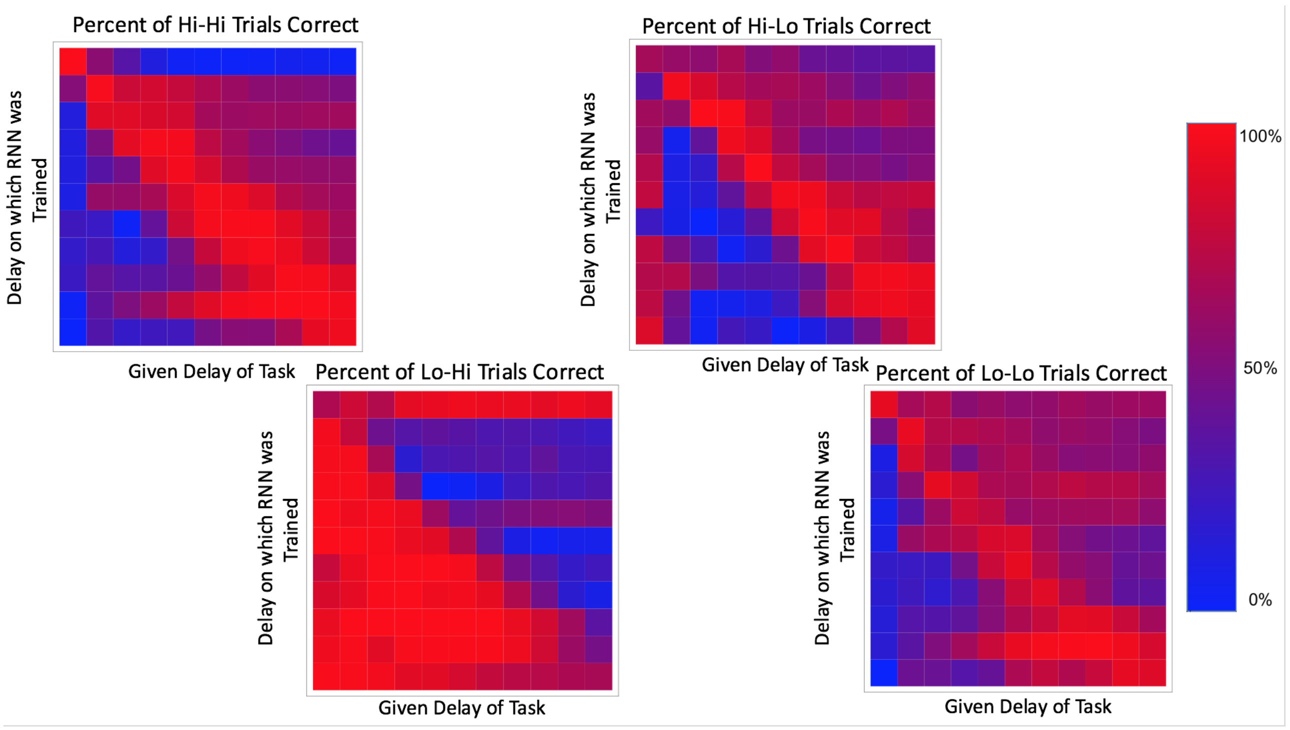
2.2 Results

Each of these families of networks trained at different delays were then tested on delay periods Δ_1, again ranging from 0 to 5 seconds in steps of .5 seconds, to see how they performed outside of the delay Δ_0 that they were trained on.

In the figure below, the rows from top to bottom represent the delays that the network was trained on, while the columns from left to right represent the given delay to the network. The color coding from blue to red represents the number of trails that the network got correct. Note that we would expect an untrained network to get *50%* of trials correct (corresponding to purple), which means blue points actually correspond to the network being *negatively tuned* to this delay period. Unsurprisingly, there is a diagonal red line when the given delay is the delay the network was trained on. Moreover, this chart seems very symmetric, with not much skew to either side of the diagonal.



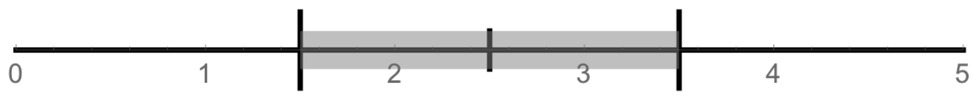
Delving further into this, however, we see surprising asymmetries in how various networks can successfully recognize Hi Hi trials (i.e. when both frequencies are high) versus Lo-Hi and the other two possible trial combinations:



Especially in the Hi-Lo (resp. Lo-Hi) trials, we see significant bias that networks trained on a given delay are very good at correctly guessing for longer (resp. shorter) durations. For the Hi-Hi and Lo-Lo trials, we also see a high tendency to be good at predicting pulses when the delay is increased rather than decreased. Also take note of the blue regions where there is actually negative correlation (the network does consistently worse than it would by just random-guessing).

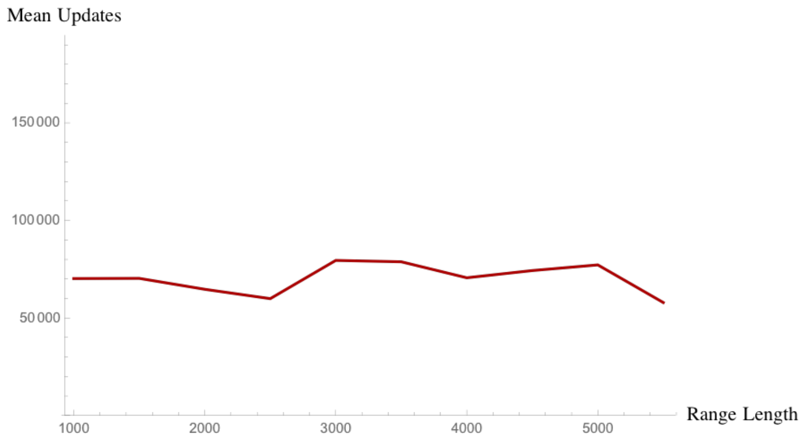
2.3 The Task of Variant Range

We also trained a set of networks in a different way. Rather than having the pulses appear exactly at a given delay away from the first one, we allowed the pulses to appear within some random window around 2.5 seconds.



The size of this window depends on the range variable R, going from 0 (exactly at 2.5) to 5 (pulse can be anywhere from 0 to 5 seconds). That is, for range R, the delay is uniformly distributed as a random variable over $[2.5 - R/2, 2.5 + R/2]$.

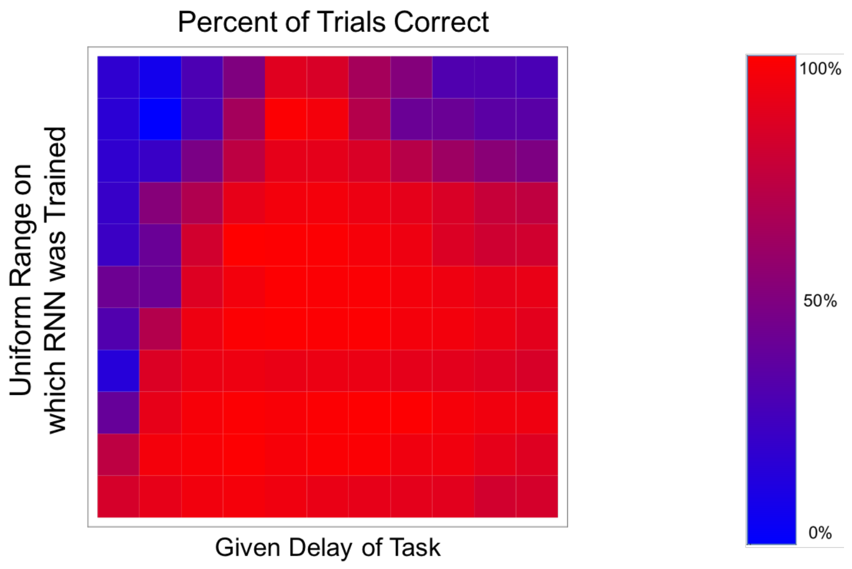
We expect that networks trained on the range variable for large ranges would take a longer time to train, but be much more accurate for all test cases regardless of range (by the very nature of how they were trained). In fact, surprisingly, the networks for larger range variables did not take more time to train for networks with small ranges. The training time was roughly constant.



The range networks trained in a number of steps essentially independent of the size of the window of randomness.

2.4 Results

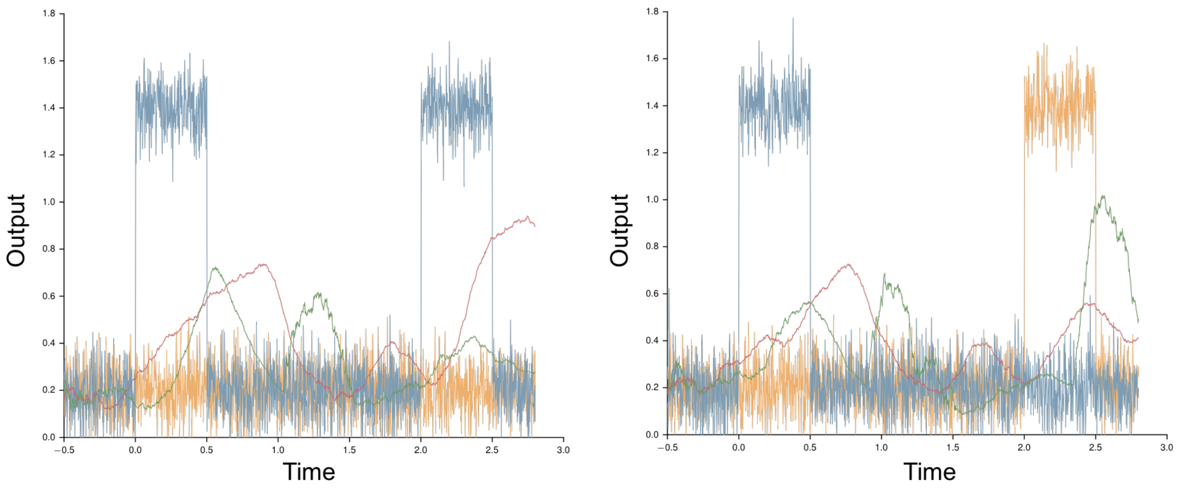
In the figure below, the rows from top to bottom correspond to increasing range. This figure is very much as expected. For range 0, it gives the same result as training a network on delay 2.5 seconds, while networks trained at range 5 discriminate correctly for almost all delays.



3 Discrimination of Inputs

3.1 The Task

Now we turn our attention to the ability of a network to discriminate between two input neurons A and B. Here, the frequency of inputs doesn't matter, so long as it is not zero.

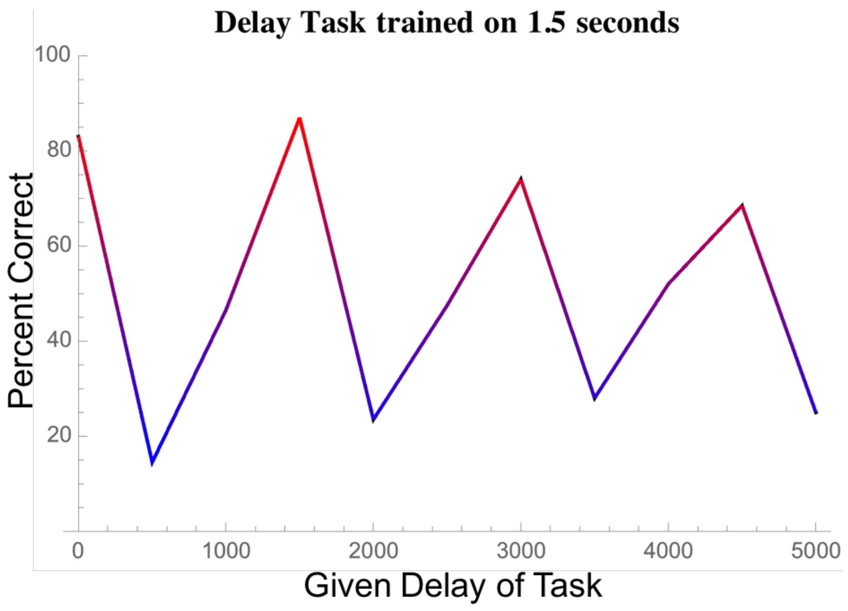


This task was much harder to train the networks to perform, and in fact very few successfully trained within a reasonable time period. No networks above 1.5 seconds trained, and only one trained at 1.5. Even with a .5 second delay, only 3 out of the five networks trained. It is worth investigating how we can fix this.

The result below is for a 1.5 second network that trained successfully. It should not be taken to be representative of all such networks for this task. It represents one point of the limited data that we have.

3.2 Results

The accuracy of the network trained on 1.5 seconds versus the delay given to the network gives an interesting oscillatory phenomenon:



This is a very unexpected emergent phenomenon that is worth further study as we understand how to train these networks to correctly discriminate between inputs. The network appears to be *negatively tuned* for delay periods at $0.5, 2, 3.5$, and 5 seconds, and positively tuned at the integer multiples of 1.5 seconds. We also see slight decay in this graph as the delay period increases.

This network appears to have some underlying structure that allows for surprisingly slow oscillation in the output neurons relative to the timescale of the neural units. It is worth studying how such slow oscillation is generated.

4 Future Steps

During the next term, the goal will first be to understand what parameters and connectivity make training easier, or even feasible. In addition to this, we would like to run more analysis on these networks, beyond just how well they generalize to tasks outside their training range. What further analysis can be done to gain understanding of the dependence between initial parameters and resulting dynamics?

One particularly interesting phenomenon that we observed was that the ability of networks to train was heavily dependent on how the input information was fed into them. There is a big difference in discriminating between Hi/Lo frequency from one input neuron vs. discriminating between input neuron A and input neuron B. Why do such disparities in training aptitude stem from simple changes of input information?

On the computational side, we'd like to speed up the training, for the sake of practicality. We will be working on enabling GPU compatibility in the next few days, and hopefully this will give us a concrete speedup in training. It is also worth investigating frameworks such as OpenMP for speeding up training through parallelization.

References

[1] Juan C. Cuevas-Tello, Manuel Valenzuela-Rendón, and Juan Arturo Nolazco-Flores. A tutorial on deep neural networks for intelligent systems. *CoRR*, abs/1603.07249, 2016.

[2] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[3] Jing Li, Ji-hang Cheng, Jing-yuan Shi, and Fei Huang. *Advances in Computer Science and Information Engineering: Volume 2*, chapter Brief Introduction of Back Propagation (BP) Neural Network Algorithm and Its Improvement, pages 553–558. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[4] H. Francis Song, Guangyu R. Yang, and Xiao-Jing Wang. Training excitatory-inhibitory recurrent neural networks for cognitive tasks: A simple and flexible framework. *PLoS Comput Biol*, 12(2):1–30, 02 2016.