Neuron, Volume *63*

Supplemental Data

Generating Coherent Patterns of Activity from Chaotic Neural Networks

David Sussillo and L.F. Abbott

Supplementary Material

FORCE Learning without RLS

It is possible to implement FORCE learning without requiring the full machinery of the recursive least squares (RLS) algorithm. To do this, we use the weight modification rule

$$\mathbf{w}(t) = \mathbf{w}(t-\Delta t) - \eta(t) e_{\_}(t)\mathbf{r}(t) \tag{1}$$

where, as in RLS,

$$e_{\_}(t) = \mathbf{w}^{\mathrm{T}}(t-\Delta t)\mathbf{r}(t) - f(t)\,. \tag{2}$$

Equation 1 is a basic delta rule with a time dependent scalar learning rate, rather than the matrix learning rate of RLS. Algorithms of this form converge more slowly than RLS and cannot achieve as complex outputs, but they are simpler and more biologically plausible. In particular, they have the advantage, from a biological perspective, of being synapse autonomous, meaning that all the information needed to compute a modification is present at the site of the synapse, with the exception of the global error signal.

The questions for the modification rule of equation 1 is how to choose the time-varying learning rate $\eta(t)$. We begin the analysis of this issue with an aside about FORCE learning that is instructive, although it does not lead to the learning rate we use. The task of making the output $z = f$, for some target function f, can be solved trivially if we are willing to use time-dependent readout weights. The readout weight vector $\mathbf{w}(t) = f(t)\mathbf{r}(t)/\mathbf{r}^{\mathrm{T}}(t)\mathbf{r}(t)$ produces the output $z(t) = \mathbf{w}^{\mathrm{T}}(t)\mathbf{r}(t) = f(t)$ ($\mathbf{r}^{\mathrm{T}}\mathbf{r}$ is almost always of order N but, if it happens to go to zero, a small positive constant can be added to the denominator of the expression for $\mathbf{w}$). Thus, the difficult problem is not finding a set of weights that produce the target output, but generating a *time-independent* set of weights that does this. Using time-dependent weights to achieve the desired output requires an external agent, i.e. the learning procedure, to manipulate the weights appropriately. The network will only produce the desired output autonomously if we find a set of time-independent weights that do the trick. To find such time-independent weights, we notice that the equation we have given for $\mathbf{w}$ is not unique. Any set of weights of the form

$$\mathbf{w}(t) = \mathbf{q}(t) + \frac{\left(f(t) - \mathbf{q}^{\mathrm{T}}(t)\mathbf{r}(t)\right)\mathbf{r}(t)}{\mathbf{r}^{\mathrm{T}}(t)\mathbf{r}(t)}$$

for an arbitrary vector $\mathbf{q}(t)$ will produce the output $z = f$. This arbitrariness gives us the freedom to slowly vary the weights from time-dependent to time-independent values without ever introducing an error into the output. With $e_{\_}(t)$ given by equation 2, we see that if we set $\eta(t) = 1/\mathbf{r}^{\mathrm{T}}(t)\mathbf{r}(t)$, equation 1 would be equivalent to the above equation with $\mathbf{q}(t) = \mathbf{w}(t - \Delta t)$. FORCE learning is a similar procedure, but instead of following this error-free trajectory, it follows a nearby path through readout weight space that allows for non-zero, but small, deviations in the output. Allowing for such small deviations is essential for the stability of the final network.

Given that we do not want to set $\eta(t) = 1/\mathbf{r}^{\mathrm{T}}(t)\mathbf{r}(t)$, we must choose another time dependence for the learning rate. There are many potential forms for $\eta(t)$, but we use the equation

$$\tau\frac{d\eta}{dt} = \eta\left(-\eta + \frac{|e|^{\gamma}}{\tau}\right), \tag{3}$$

where γ is a positive number and τ is the time constant of the network neurons. In practice, $\gamma = 1$, 3/2 or 2 all appear to be effective, though we primarily use $\gamma = 3/2$. The extra factor of η multiplying the right side of equation 3 serves to slow the changes in η down as the error diminishes.

Figure S1 shows an example of this form of FORCE learning (equations 1 and 3) being used to train an initially chaotic network to produce a periodic function. The learning rate is reduced during training from an initial value of $\eta = 2 \times 10^{-3}/\tau$ to a final value of $3.3 \times 10^{-5}/\tau$ (figure S1B), at which point learning is stopped (with the results shown to the right of the vertical line in the right panel of figure S1A). During the entire learning process, $z(t)$ closely matches $f(t)$, as it does after learning. The total learning time is $60{,}000\tau$ which, for $\tau = 10$ ms, is equivalent to 600 seconds.

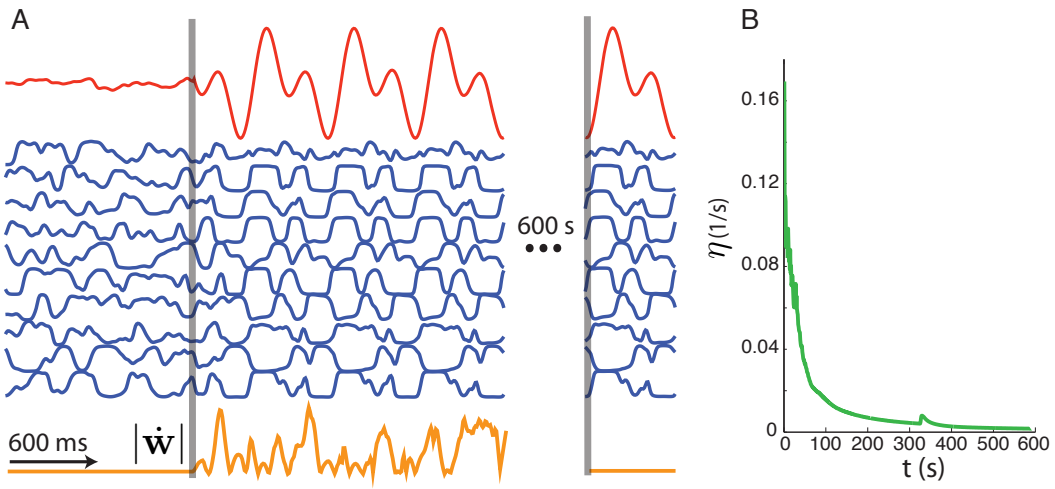

Figure S1: A) An example of non-RLS FORCE Learning. Activity is initially chaotic (left of vertical line in left panel). Learning lasted 600 s (right of vertical line in left panel), after which (right panel) the network output (red) matched the target function with static readout weights (orange). Although we only show a small time slice, this post-learning output is periodic, continues indefinitely and matches the target. B) The learning rate during the example of A.

Learning with Two Feedack Loops

For figure 2J, we added a second feedback loop to induce a fixed point that initialized the network prior to it generating a one-shot output sequence. Here is the learning procedure used for this example. We begin the training process by activating readout unit 1 and

applying FORCE learning to the weights onto both readout units. The target function for readout unit 1 is simply the constant value 1. The target function for readout unit 2 is, at this point, the initial value of the one-shot sequence we ultimately want to learn (the value of the red trace at the left blue arrow in figure 2J). After a brief learning period, the weights attain a value that puts the entire network into a fixed point state, with constant activities for all the network neurons, a constant value for output 1, and a constant value for output 2 equal to the initial value of its target sequence. This state is henceforth achieved whenever both readout units are activated. We then train the network to produce the one-shot sequence by first activating and then deactivating unit 1, and FORCE training the readout weights of unit 2 with the desired one-shot target function. This two-step process (activating and inactivating unit 1 while modifying the weights of unit 2) is repeated until the network can produce the desired output in this way without requiring weight modification. From then on, the one-shot sequence can be generated by activating readout unit 1 to briefly put the network into a fixed-point state, and then inactivating it to generate the sequence. After the learned sequence is completed, the output of unit 2 becomes chaotic, although occasionally the learned sequence or pieces of it may be generated spontaneously.

Learning Rules for Feedback and Generator Networks

The application of the RLS FORCE learning rules to synapses within the feedback or generator networks is identical to its application to the readout weights, except that the firing rates presynaptic to the network neuron being modified are used rather than the rates driving the readout unit. Here we give the equations for modifying synapses within these networks, which are conceptual simple but, notationally a bit involved.

For synapse modification within the feedback network, we use

$$J^{\rm FG}_{ai}(t) = J^{\rm FG}_{ai}(t-\Delta t) - e_{\_}(t) \sum_{j\in\mathbf{A}(a)} P^a_{ij}(t) r_j(t)\,,$$

where we use the notation $\mathbf{A}(a)$ to stand for the list of all generator neurons presynaptic to feedback neuron a. Using this notation, $\mathbf{P}^a$ is a square matrix with each dimension equal to the number of generator network neurons presynaptic to feedback neuron a, and it is updated by

$$P^a_{ij}(t) = P^a_{ij}(t-\Delta t) - \frac{\sum\limits_{k\in\mathbf{A}(a)}\sum\limits_{l\in\mathbf{A}(a)} P^a_{ik}(t-\Delta t) r_k(t) r_l(t) P^a_{lj}(t-\Delta t)}{1+\sum\limits_{k\in\mathbf{A}(a)}\sum\limits_{l\in\mathbf{A}(a)} r_k(t) P^a_{kl}(t-\Delta t) r_l(t)}\,.$$

For synapse modification within the generator network, we use

$$J^{\rm GG}_{ij}(t) = J^{\rm GG}_{ij}(t-\Delta t) - e_{\_}(t) \sum_{k\in\mathbf{B}(i)} P^i_{jk}(t) r_k(t)\,,$$

where $\mathbf{B}(i)$ is the list of all generator neurons presynaptic to generator neuron i. Using this notation, $\mathbf{P}^i$ is a square matrix with each dimension equal to the number of generator

network neurons presynaptic to neuron i and is updated by

$$P^i_{jk}(t) = P^i_{jk}(t-\Delta t) - \frac{\sum\limits_{l\in\mathbf{B}(i)}\sum\limits_{m\in\mathbf{B}(i)} P^i_{jl}(t-\Delta t)r_l(t)r_m(t)P^i_{mk}(t-\Delta t)}{1+\sum\limits_{l\in\mathbf{B}(i)}\sum\limits_{m\in\mathbf{B}(i)} r_l(t)P^i_{lm}(t-\Delta t)r_m(t)}\,.$$

Eigenvalues of the Effective Synaptic Matrix

In the main text, figure 5, we considered the quality of training as a function of the synaptic scaling parameter g. Given that training performance increases as g increases (up to a point), it is instructive to view the eigenvalues of the full synaptic matrix both before and after training for different values of g. Here, we are discussing the feedback architecture of figure 1A, and by full synaptic matrix we mean the following. The equation for the generator network neurons,

$$\tau\frac{dx_i}{dt} = -x_i + g\sum_{j=1}^{N_{\rm G}} J^{\rm GG}_{ij}r_j + J^{{\rm G}z}_i z \tag{4}$$

and the definition of the output

$$z(t) = \mathbf{w}^{\rm T}\mathbf{r}(t) \tag{5}$$

can be combined to give

$$\tau\frac{dx_i}{dt} = -x_i + \sum_{j=1}^{N_{\rm G}} J^{\rm eff}_{ij}r_j \tag{6}$$

where

$$J^{\rm eff}_{ij} = gJ^{\rm GG}_{ij} + J^{{\rm G}z}_i w_j \tag{7}$$

with w_j the $j^{\rm th}$ component of $\mathbf{w}$. The eigenvalues we show (figure S2) and discuss here are those of the matrix $J^{\rm eff}$.

We trained two networks, one with $g = 0.8$ and one with $g = 1.5$, to generate the output function of figure 2D, which is a sum of 4 sinusoids. In panel A of figure S2, the eigenvalues of $J^{\rm eff}$ are shown for the network with $g = 0.8$. Note that after training (red circles) a few eigenvalues have moved a large distance away from the pre-training eigenvalues (blue circles). On the other hand, for a network with $g = 1.5$ (figure S2, panel B) there is little distance between the pre- and post-training eigenvalues. Both before and after learning, the eigenvalues totally surround the frequencies of the target function. Also, many eigenvalues reside in the linearly unstable part of the complex plane, meaning their real parts are greater than 1, both before and after training. It is evident from the two plots that the solutions found by the two networks are qualitatively different.

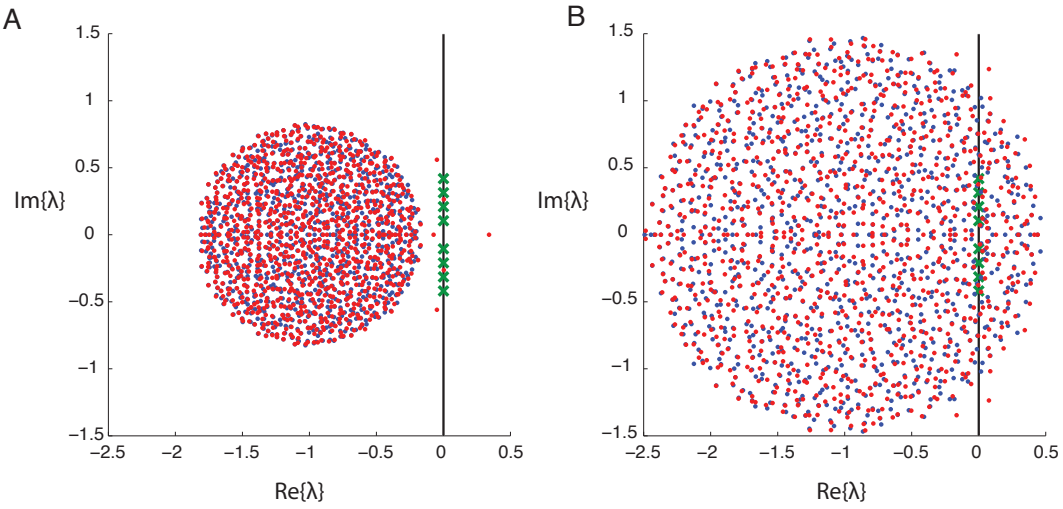

Figure S2: Eigenvalues of the effective synaptic matrices before and after learning. The target function is the sum of 4 sinusoids shown in figure 2D. Each of the four frequencies of these sinusoids is denoted by a green 'X' located at the appropriate position on the imaginary axis of the complex plane. Red circles are eigenvalues after training and blue circles before training. The vertical line indicates a real part of 1. To the right of this line, eigenvalues correspond to modes that are unstable for a linearization of the network around the point $\mathbf{x} = 0$. A) The eigenvalues of the synaptic matrices of a network with $g = 0.8$, which gives a quiescent behavior in the absence of input. The effect of training is to move a few eigenvalues a long way. In particular, a single eigenvalue moves into the linearly unstable part of the imaginary plane, and a couple of eigenvalues move in and around the frequency range of the 4 sinusoids of the target function. B) The eigenvalues of a network with $g = 1.5$. Eigenvalues move by small amounts, and the relationship between the 4 frequencies of the target function and the trained eigenvalues is not obvious.