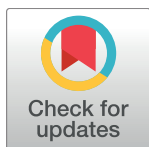# full-FORCE: A target-based method for training recurrent networks

Brian DePasquale[1¤]*, Christopher J. Cueva[1], Kanaka Rajan[2,3], G. Sean Escola[4], L. F. Abbott[1,5]

1 Department of Neuroscience, Zuckerman Institute, Columbia University, New York, NY, United States of America, **2** Joseph Henry Laboratories of Physics and Lewis-Sigler Institute for Integrative Genomics, Princeton University, Princeton, NJ, United States of America, **3** Friedman Brain Institute, Icahn School of Medicine at Mount Sinai, One Gustave L. Levy Place, New York, NY, United States of America, **4** Department of Psychiatry, Columbia University College of Physicians and Surgeons, New York, NY, United States of America, **5** Department of Physiology and Cellular Biophysics, Columbia University College of Physicians and Surgeons, New York, NY, United States of America

¤ Current address: Princeton Neuroscience Institute, Princeton University, Princeton, NJ, United States of America

* depasquale@princeton.edu

Abstract

Trained recurrent networks are powerful tools for modeling dynamic neural computations. We present a target-based method for modifying the full connectivity matrix of a recurrent network to train it to perform tasks involving temporally complex input/output transformations. The method introduces a second network during training to provide suitable "target" dynamics useful for performing the task. Because it exploits the full recurrent connectivity, the method produces networks that perform tasks with fewer neurons and greater noise robustness than traditional least-squares (FORCE) approaches. In addition, we show how introducing additional input signals into the target-generating network, which act as task hints, greatly extends the range of tasks that can be learned and provides control over the complexity and nature of the dynamics of the trained, task-performing network.

Introduction

A principle focus in systems and circuits neuroscience is to understand how the neuronal representations of external stimuli and internal intentions generate actions appropriate for a particular task. One fruitful approach for addressing this question is to construct (or "train") model neural networks to perform analogous tasks. Training a network model is done by adjusting its parameters until it generates desired "target" outputs in response to a given set of inputs. For layered or recurrent networks, this is difficult because no targets are provided for the "interior" (also known as hidden) units, those not directly producing the output. This is the infamous credit-assignment problem. The most widely used procedure for overcoming this challenge is stochastic gradient-decent using backpropagation (see, for example, [1]), which uses sequential differentiation to modify interior connection weights solely on the basis of the discrepancy between the actual and target outputs. Although enormously successful, this

procedure is no panacea, especially for the types of networks and tasks we consider [2]. In particular, we construct continuous-time networks that perform tasks where inputs are silent over thousands of model integration time steps. Using backpropagation through time [3] in such cases requires unfolding the network dynamics into thousands of effective network layers and obtaining gradients during time periods during which, as far as the input is concerned, nothing is happening. In addition, we are interested in methods that extend to spiking network models [4]. As an alternative to gradient-based approaches, we present a method based on deriving targets not only for the output but also for interior units, and then using a recursive least-squares algorithm [5] to fit the activity of each unit to its target.

Target- rather than backpropagation-based learning has been proposed for feedforward network architectures [6]. Before discussing a number of target-based methods for recurrent networks and presenting ours, we describe the network model we consider and define its variables and parameters. We use recurrently connected networks of continuous variable "firing-rate" units that do not generate action potentials (although see [4]). The activity of an N-unit model network (Fig 1a) is described by an N-component vector $\mathbf{x}$ that evolves in continuous time according to

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{J}H(\mathbf{x}) + \mathbf{u}_{\text{in}}f_{\text{in}}(t) \,, \tag{1}$$

where τ sets the time scale of the network dynamics (for the examples we show, τ = 10 ms). H is a nonlinear function that maps the vector of network activities $\mathbf{x}$ into a corresponding vector of "firing rates" $H(\mathbf{x})$ (we use $H(\mathbf{x}) = \tanh(\mathbf{x})$). $\mathbf{J}$ is an $N \times N$ matrix of recurrent connections between network units. An input $f_{\text{in}}(t)$ is provided to the network units through a vector of input weights $\mathbf{u}_{\text{in}}$. The output of the network, $z(t)$, is defined as a sum of unit firing rates weighted by a vector $\mathbf{w}$,

$$z(t) = \mathbf{w}^{\text{T}}H(\mathbf{x}(t)) \,. \tag{2}$$

Tasks performed by this network are specified by maps between a given input $f_{\text{in}}(t)$ and a desired or target output $f_{\text{out}}(t)$. Successful performance of the task requires that $z(t) \approx f_{\text{out}}(t)$ to a desired degree of accuracy.

A network is trained to perform a particular task by adjusting its parameters. In the most general case, this amounts to adjusting $\mathbf{J}$, $\mathbf{w}$ and $\mathbf{u}_{\text{in}}$, but we will not consider modifications of $\mathbf{u}_{\text{in}}$. Instead, the elements of $\mathbf{u}_{\text{in}}$ are chosen independently from a uniform distribution
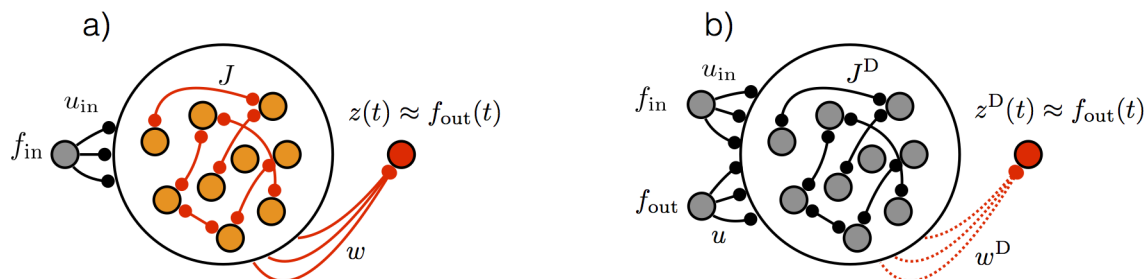
Fig 1. Network architecture. (a) Task-performing network. The network receives $f_{\text{in}}(t)$ as an input. Training modifies the elements of $\mathbf{J}$ and $\mathbf{w}$ so that the network output $z(t)$ matches a desired target output function $f_{\text{out}}(t)$. (b) Target-generating network. The network receives $f_{\text{out}}(t)$ and $f_{\text{in}}(t)$ as inputs. Input connections $\mathbf{u}$, $\mathbf{u}_{\text{in}}$ and recurrent connections $\mathbf{J}^{\text{D}}$ are fixed and random. To verify that the dynamics of the target-generating network are sufficient for performing the task, an optional linear projection of the activity, $z^{D}(t)$, can be constructed by learning output weights $\mathbf{w}^{\text{D}}$, but this is a check, not an essential step in the algorithm.

https://doi.org/10.1371/journal.pone.0191527.g001

between -1 and 1 and left fixed. The cost function being minimized is

$$C_{\mathbf{w}} = \left\langle (z(t) - f_{\text{out}}(t))^2 \right\rangle, \tag{3}$$

where the angle brackets denote an average over time during a trial and training examples. The credit assignment problem discussed above arises because we only have a target for the output z, namely f_{out}, and not for the vector $\mathbf{x}$ of network activities. Along with backpropagation, a number of approaches have been used to train recurrent networks of this type. A number of of these involve either ways of circumventing the credit assignment problem or methods for deducing targets for $\mathbf{x}(t)$.

In liquid- or echo-state networks [7–9], no internal targets are required because modification of the internal connections $\mathbf{J}$ is avoided entirely. Instead, modification only involves the output weights $\mathbf{w}$. In this case, minimizing $C_{\mathbf{w}}$ is a simple least-squares problem with a well-known solution for the optimal $\mathbf{w}$. The price paid for this simplicity in learning, however, is limited performance from the resulting networks.

An important next step [10] was based on modifying the basic network eq 1 by feeding the output back into the network through a vector of randomly chosen weights $\mathbf{u}$,

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{J}H(\mathbf{x}) + \mathbf{u}_{\text{in}}f_{\text{in}}(t) + \mathbf{u}z(t). \tag{4}$$

Because $z = \mathbf{w}^{\text{T}}H(x)$, this is equivalent to replacing the matrix of connections $\mathbf{J}$ in Eq 1 by $\mathbf{J} + \mathbf{u}\mathbf{w}^{\text{T}}$. Learning is restricted, as in liquid- and echo-state networks, to modification of the output weight vector $\mathbf{w}$ but, because of the additional term $\mathbf{u}\mathbf{w}^{\text{T}}$, this also generates a limited modification in the effective connections of the network. Modification of the effective connection matrix is limited in two ways; it is low rank (rank one in this example), and it is tied to the modification of the output weight vector. Nevertheless, when combined with a recursive least-squares algorithm for minimizing $C_{\mathbf{w}}$, this process, known as FORCE learning, is an effective way to train recurrent networks [11].

Although the FORCE approach greatly expands the capabilities of trained recurrent networks, it does not take advantage of the full recurrent connectivity because of the restrictions on the rank and form of the modifications it implements. Some studies have found that networks trained by FORCE to perform complex "real-world" problems such as speech recognition require many more units to match the performance of networks trained by gradient-based methods [12]. In addition, because of the reliance of FORCE on random connectivity, the activity of the resulting trained networks can be overly complex compared, for example, to experimental recordings [13].

A suggestion has been made for extending the FORCE algorithm to permit more general internal learning [11, 14]. The idea is to use the desired output f_{out} to generate targets for every internal unit in the network. In this approach, the output is not fed back into the network, which is thus governed by Eq 1 not Eq 4. Instead a random vector $\mathbf{u}$ is used to generate targets, and $\mathbf{J}$ is adjusted by a recursive least-squares algorithm that minimizes

$$C_{\mathbf{J}}^{\text{eF}} = \left\langle |\mathbf{J}H(\mathbf{x}(t)) - \mathbf{u}f_{\text{out}}(t)|^2 \right\rangle. \tag{5}$$

Although this "extended" FORCE procedure can produce functioning networks, minimizing the above cost function is a rather unusual learning goal, If learning could set $C_{\mathbf{J}}^{\text{eF}} = 0$, the effective equation of the network would be $\tau d\mathbf{x}/dt = -\mathbf{x} + \mathbf{u}f_{\text{out}}(t) + \mathbf{u}_{\text{in}}f_{\text{in}}(t)$. This equation is incompatible with the output $z(t)$ being equal to $f_{\text{out}}(t)$ because $f_{\text{out}}(t)$ cannot, in general, be constructed from a low-pass filtered version of itself. Thus, the success of this scheme relies on

failing to make C_J^{eF} too small, but succeeding enough to assure that the target output is a partial component in the response of each unit.

Laje & Buonomano [15] proposed a scheme that uses a second "target-generating" network to produce targets for the activities of the network being constructed. They reasoned that the rich dynamics of a randomly connected network operating in the chaotic regime [16] would provide a general basis for many dynamic tasks, but they also noted that the sensitivity of chaotic dynamics to initial conditions and noise ruled out chaotic networks as a source of this basis (see also [17]). To solve this problem, they used the activities of the chaotic target-generating network, which we denote as $\mathbf{x}^{chaos}(t)$, as targets for the actual network they wished to construct (which we call the "task-performing" network). They adjusted $\mathbf{J}$ to minimized the cost function

$$C_J^{LB} = \left\langle |H(\mathbf{x}(t)) - H(\mathbf{x}^{chaos}(t))|^2 \right\rangle . \tag{6}$$

After learning, the task-performing network matches the activity of the target-generating network, but it does so in a non-chaotic "stable" way, alleviating the sensitivity to initial conditions and noise of a truly chaotic system. Once this stabilization has been achieved, the target output is reproduced as accurately as possible by adjusting the output weights $\mathbf{w}$ to minimize the cost function 3.

Like the approach of Laje & Buonomano [15], our proposal uses a second network to generate targets, but this target-generating network operates in a driven, non-chaotic regime. Specifically, the target-generating network is a randomly connected recurrent network driven by external input that is strong enough to suppress chaos [18]. The input to the target-generating network is proportional to the target output $f_{out}(t)$, which gives our approach some similarities to the extended FORCE idea discussed above [11, 14]. However, in contrast to that approach, the goal here is to minimize the cost function as much as possible rather than relying on limited learning. Because our scheme allows general modifications of the full connection matrix $\mathbf{J}$, but otherwise has similarities to the FORCE approach, we call it full-FORCE.

In the following, we provide a detailed description of full-FORCE and illustrate its operation in a number of examples. We show that full-FORCE can construct networks that successfully perform tasks with fewer units and more noise resistance than networks constructed using the FORCE algorithm. Networks constructed by full-FORCE have the desirable property of degrading smoothly as the number of units is decreased or noise is increased. We discuss the reasons for these features. Finally, we note that additional signals can be added to the target-generating network, providing "hints" to the task-performing network about how the task should be performed [19, 20]. Introducing task hints greatly improves network learning and significantly extends the range of tasks that can be learned. It also allows for the construction of models that span the full range from the more complex dynamics inherited from random recurrent networks to the often simple dynamics of "hand-built" solutions.

Materials and methods

The full-FORCE algorithm

As outlined in the introduction, the full-FORCE approach involves two different networks, a target-generating network used only during training and a task-performing network that is the sole final product of the training procedure. The target-generating network is a random recurrent network that is not modified by learning. We denote the variables and parameters of the target-generating network by a superscript D standing for "driven". This is because the target-

generating network receives the target output signal as an input that takes part in driving its activity.

The activities in the N-unit target-generating network (Fig 1b), described by the vector $\mathbf{x}^{\mathrm{D}}$, are determined by

$$\tau \frac{d\mathbf{x}^{\mathrm{D}}}{dt} = -\mathbf{x}^{\mathrm{D}} + \mathbf{J}^{\mathrm{D}}H(\mathbf{x}^{\mathrm{D}}) + \mathbf{u}f_{\mathrm{out}}(t) + \mathbf{u}_{\mathrm{in}}f_{\mathrm{in}}(t) \, . \tag{7}$$

The final term in this equation is identical to the input term in Eq 1. The connection matrix $\mathbf{J}^{\mathrm{D}}$ is random with elements chose i.i.d. from a Gaussian distribution with zero mean and variance g^2/N. We generally use a g value slightly greater than 1 (for all of the examples we present here, $g = 1.5$), meaning that, in the absence of any input, the target-generating network would exhibit chaotic activity [16]. However, importantly, this chaotic activity is suppressed by the two inputs included in Eq 7. The new input term, not present in Eq 1, delivers the target output $f_{\mathrm{out}}(t)$ *into* the network through a vector of weights $\mathbf{u}$. The components of $\mathbf{u}$, like those of $\mathbf{u}_{\mathrm{in}}$, are chosen i.i.d. from a uniform distribution between -1 and 1. For all of the examples we present, the numbers of units in the driven and task-generating networks are the same, but we consider the possibility of relaxing this condition in the discussion.

The idea behind the target-generating network described by Eq 7 is that we want the final task-performing network, described by Eq 1, to mix the dynamics of its recurrent activity with signals corresponding to the target output $f_{\mathrm{out}}(t)$. If this occurs, it seems likely that a linear readout of the form 2 should be able to extract the target output from this mixture. Through Eq 7, we assure that the activities of the target-generating network reflect exactly such a mixture. For the target-generating network, the presence of signals related to $f_{\mathrm{out}}(t)$ is imposed by including this function as a driving input. This input is not present in Eq 1 describing the task-performing network; it must generate this signal internally. Thus, learning in full-FORCE amounts to modifying $\mathbf{J}$ so that the task-performing network generates signals related to the target output *internally* in a manner that matches the mixing that occurs in the target-generating network when $f_{\mathrm{out}}(t)$ is provided *externally*. More explicitly, we want the combination of internal and external signals in the target-generating network, $\mathbf{J}^{\mathrm{D}} H(\mathbf{x}^{\mathrm{D}}) + \mathbf{u}f_{\mathrm{out}}(t)$, to be matched by the purely internal signal in the task-performing network, $\mathbf{J}H(\mathbf{x})$. This is achieved by adjusting $\mathbf{J}$ to minimize the cost function

$$C_{\mathbf{J}}^{\mathrm{fF}} = \left\langle \left| \mathbf{J}H(\mathbf{x}(t)) - \mathbf{J}^{\mathrm{D}}H(\mathbf{x}^{\mathrm{D}}(t)) - \mathbf{u}f_{\mathrm{out}}(t) \right|^2 \right\rangle \, . \tag{8}$$

This is done, as in FORCE, by using the recursive least-squares algorithm. Before learning, $\mathbf{J}$ is not required to be the same as $\mathbf{J}^{\mathrm{D}}$ (for the examples we show, $\mathbf{J} = 0$ before learning).

Defining the full-FORCE error at time t as

$$e(t) = \mathbf{J}(t - \Delta t)H(\mathbf{x}(t)) - \mathbf{J}^{\mathrm{D}}H(\mathbf{x}^{\mathrm{D}}(t)) - \mathbf{u}f_{\mathrm{out}}(t) \, , \tag{9}$$

the RLS algorithm updates $\mathbf{J}$ according to

$$\mathbf{J}(t) = \mathbf{J}(t - \Delta t) - e(t)^{\mathrm{T}}\mathbf{P}(t)H(\mathbf{x}(t)) \, , \tag{10}$$

where $\mathbf{P}(t)$ is a running estimate of the inverse of the correlation matrix of $H(\mathbf{x}(t))$. $\mathbf{P}(t)$ is updated at each RLS step according to

$$\mathbf{P}(t) = \mathbf{P}(t - \Delta t) - \frac{\mathbf{P}(t - \Delta t)H(\mathbf{x}(t))H(\mathbf{x}(t))^{\mathrm{T}}\mathbf{P}(t - \Delta t)}{1 + H(\mathbf{x}(t))^{\mathrm{T}}\mathbf{P}(t - \Delta t)H(\mathbf{x}(t))} \, . \tag{11}$$

$\mathbf{P}$ is initialized to $\mathbf{P}(0) = \mathbf{I}/\alpha$, where $\mathbf{I}$ is the identity matrix. The parameter α, which can

potentially take on values over several orders of magnitude, sets the scale of an ℓ_2-regularization term in the least squares cost function and acts as an effective learning rate. Adjusting α can improve learning. In all of our examples $\alpha = 1$. Times between learning updates, corresponding to Δt in Eq 9, are chosen randomly (with an average of 2 ms) to avoid redundant sampling in periodic tasks.

The primary assumption underlying full-FORCE is that the target output can be extracted by a linear readout from the mixture of internal and external signals within the activities of the target-generating network once they are transferred to the task-performing network. It is important to note that this assumption can be checked (although this is not a required step in the algorithm) by adding a readout to the target-generating network (Fig 1b) and determining whether it can extract the desired output. In other words, it should be possible to find a set of weights $\mathbf{w}^D$ (this can be done by recursive or batch least-squares) for which $(\mathbf{w}^D)^T H(\mathbf{x}^D) = \mathbf{z}^D(t) \approx f_{\text{out}}(t)$ to a desired degree of accuracy. This readout, combined with the fact that $f_{\text{out}}(t)$ is an input to the target-generating network, means that we are requiring the target-generating network to act as an auto-encoder of the signal $f_{\text{out}}(t)$ despite the fact that it is a strongly-coupled nonlinear network.

In recurrent networks, learning must not only produce activity that supports the desired network output, it must also suppress any instabilities that might cause small fluctuations in the network activity to grow until they destroy performance. Indeed, stability is the most critical and difficult part of learning in recurrent networks with non-trivial dynamics [11, 15, 17]. As we will show, the stabilization properties of full-FORCE are one of its attractive features.

The networks we use often, at least initially, display chaotic activity. Sometimes, especially in cases where the input consists of temporally separated pulses, we shift the target output away from zero. This constant offset acts to suppress the chaotic activity.

Results

Differences between full-FORCE and FORCE learning

The connection matrix $\mathbf{J}$ that minimizes the cost function of Eq 8 is given by

$$\mathbf{J} = (\mathbf{J}^D \langle H(\mathbf{x}^D) H(\mathbf{x}^T) \rangle + \mathbf{u} \langle f_{\text{out}} H(\mathbf{x}^T) \rangle) \langle H(\mathbf{x}) H(\mathbf{x}^T) \rangle^{-1}. \tag{12}$$

If the activities of the target-generating and task-performing networks were the same, $\mathbf{x}^D = \mathbf{x}$, and the output of the task-performing network was perfect, $\mathbf{w}^T H(\mathbf{x}) = f_{\text{out}}$, Eq 12 would reduce to $\mathbf{J} = \mathbf{J}^D + \mathbf{u}\mathbf{w}^T$, which is exactly what the FORCE algorithm would produce if its original recurrent connection matrix was $\mathbf{J}^D$. Of course, we cannot expect the match between the two networks and between the actual and desired outputs to be perfect, but this result might suggest that if these matches are close, $\mathbf{J}$ will be close to $\mathbf{J}^D + \mathbf{u}\mathbf{w}^T$ and full-FORCE will be practically equivalent to FORCE. This expectation is, however, incorrect.

To clarify what is going on, we focus on the first term in Eq 12 (the term involving $\mathbf{J}^D$; the argument concerning the second term is virtually identical). This term involves the expression $\langle H(\mathbf{x}^D) H(\mathbf{x}^T) \rangle \langle H(\mathbf{x}) H(\mathbf{x}^T) \rangle^{-1}$, which obviously reduces to the identity matrix if $\mathbf{x} = \mathbf{x}^D$ (and thus the term we are discussing reduces to $\mathbf{J}^D$). Nevertheless, this expression can be quite different from the identity matrix when $\mathbf{x}$ is close to but not equal to $\mathbf{x}^D$, which is what happens in practice. To see why, it is useful to use a basis in which the correlation matrix of task-performing network firing rates is diagonal, which is the principle component (PC) basis. In this basis, the difference between $\langle H(\mathbf{x}^D) H(\mathbf{x}^T) \rangle \langle H(\mathbf{x}) H(\mathbf{x}^T) \rangle^{-1}$ and the identity matrix is expressed as a sum over PCs. The magnitude of the term in this sum corresponding to the nth PC is equal to the projection of $H(\mathbf{x}^D) - H(\mathbf{x})$ onto that PC divided by the square root of its PC eigenvalue λ_n. This means that deviations in the projection of the difference between the rates of the target-

generating and task-performing networks along the nth PC only have to be of order $\sqrt{\lambda_n}$ to pull $\langle H(\mathbf{x}^D) H(\mathbf{x}^T) \rangle \, \langle H(\mathbf{x}) H(\mathbf{x}^T) \rangle^{-1}$ significantly away from the identity matrix. The spectrum of eigenvalues of the correlation matrix of the activities in the networks we are discussing falls exponentially with increasing n [21]. Therefore, small deviations between $H(\mathbf{x}^D)$ and $H(\mathbf{x})$ can generate large differences between $\mathbf{J}$ and $\mathbf{J}^D$. We illustrate these differences in a later section by examining the eigenvalue spectrum of $\mathbf{J}$ and $\mathbf{J}^D + \mathbf{u}\mathbf{w}^T$ for two networks trained for a specific example task.

According to the above analysis, the deviations that cause the results of FORCE and full-FORCE to differ lie in spaces spanned by PCs that account for very little of the variance of the full network activity. This might make them appear irrelevant. However, as discussed above, stabilizing fluctuations is a major task for learning in recurrent networks, and the most difficult fluctuations to stabilize are those in this space. The good stabilization properties of full-FORCE are a result of the fact that deviations aligned with PCs that account for small variances modify $\mathbf{J}$ way from $\mathbf{J}^D$.
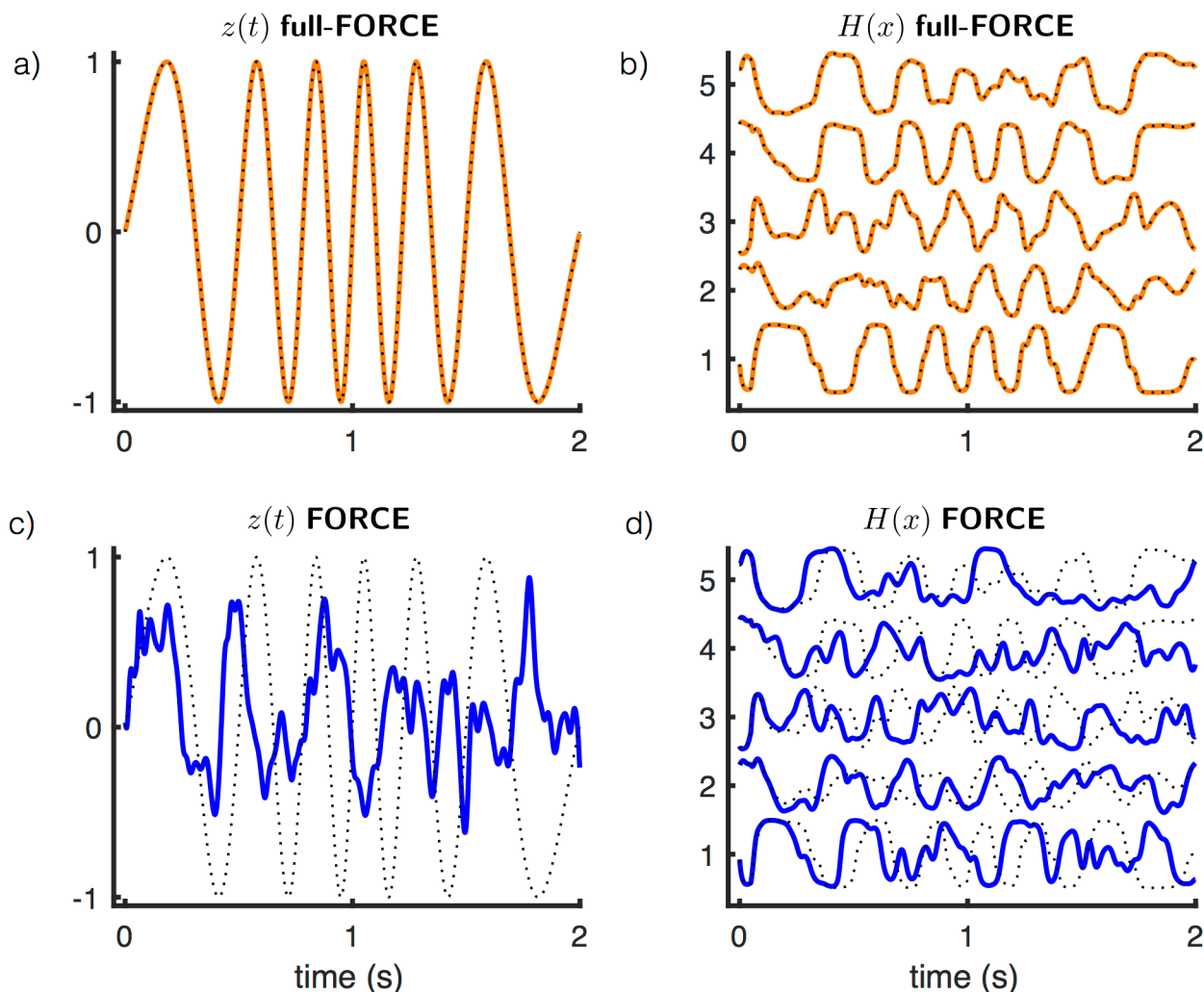
An oscillation task

To illustrate how full-FORCE works and to evaluate its performance, we considered a task in which a brief input pulse triggers a frequency-modulated oscillation in $f_{\text{out}}(t)$, and this pattern repeats periodically every 2 s. Specifically, $f_{\text{out}}(t) = \sin(\omega(t)t)$ with $\omega(t)$ increasing linearly from 2π to 6π Hz for the first half of the oscillation period, and then the signal is reflected in time around the midpoint of the period, resulting in a periodic accordian-like curve (Fig 2a, black dotted trace). This task was chosen because it is challenging but possible for recurrent network models to achieve. The input pulse was included to prevent a slow drift of the oscillation generated by the network, after training, from the target oscillation, which occurs inevitably in a fully autonomous system.

A full-FORCE-trained network of 300 units can perform this task, whereas a network of this size trained with traditional FORCE cannot (Fig 2). The random connectivity of the FORCE network was set equal to $\mathbf{J}^D$, so the target-generating network used for full-FORCE is equivalent to a teacher-forced version of the FORCE network (a network in which the target rather than the actual output is feed back). The FORCE network fails because its units cannot generate the activity that teacher-forcing demands (Fig 2d). We examined more generally how learning is affected by the number of units for both algorithms (Fig 3). Test error was computed over 50 periods of network output as the optimized value of the cost function of Eq 3 divided by the variance of f_{out}.

A network trained using full-FORCE can solve the oscillation task reliably using 200 units, whereas FORCE learning required 400 units. In addition, the performance of networks trained by full-FORCE degrades more gradually as N is decreased than for FORCE-trained networks, for which there is a more abrupt transition between good and bad performance. This is due to the superior stability properties of full-FORCE discussed above. Networks trained with full-FORCE can be inaccurate but still stable. With FORCE, instabilities are the limiting factor so networks tend to be either stable and performing well or unstable and, as a result, highly inaccurate.

To check whether long periods of learning could modify these results, we trained networks on this task using either full-FORCE or FORCE with training sets of increasing size (Fig 4). We trained both networks, using up to 100 training batches, where each batch consisted of 100 periods of the oscillation. In cases when training does not result in a successful network, long training does not appreciably improve performance for either algorithm. For both algorithms

Fig 2. Example outputs and unit activities. A network of 300 units trained with full-FORCE (a & b) and FORCE (c & d) on the oscillation task. (a) $f_{out}(t)$ (black dotted) and $z(t)$ (orange) for a network trained with full-FORCE. (b) Unit activities (orange) for 5 units from the full-FORCE-trained network compared with the target activities for these units provided by the target-generating network (black dotted). (c) $f_{out}(t)$ (black dotted) and $z(t)$ (blue) for a network trained with FORCE. (d) Unit activities (orange) for 5 units from the FORCE-trained network compared with same target activities shown in b (black dotted). Because the random matrix used in the FORCE network was $\mathbf{J}^D$, activities in a functioning FORCE network should match the activities from the target-generating network.

there appears to be a minimum required network size independent of training duration, and this size is smaller for full-FORCE than for FORCE.

Noise robustness. We examined the noise robustness of the networks we trained to perform the frequency-modulated oscillator task. Independent Gaussian white-noise was added as an input to each network unit during both learning and testing. We studied a range of network sizes and five different magnitudes of white noise with diffusion coefficient satisfying $2D = 10^{-3}, 10^{-2}, 10^{-2}, 10^{-1}$ and 10^0 /ms (Fig 5). The full-FORCE networks are more resistant to noise and their performance degrades more continuously as a function of both N (similar to what is seen in Fig 3) and the noise level. The robustness to noise for full-FORCE is qualitatively similar to results observed for networks based on stabilized chaotic activity [15].

Curiously, over a region of N values when noise is applied, the performance of the FORCE-trained networks *decreases* with increasing N (Fig 5b). This happens because partial learning
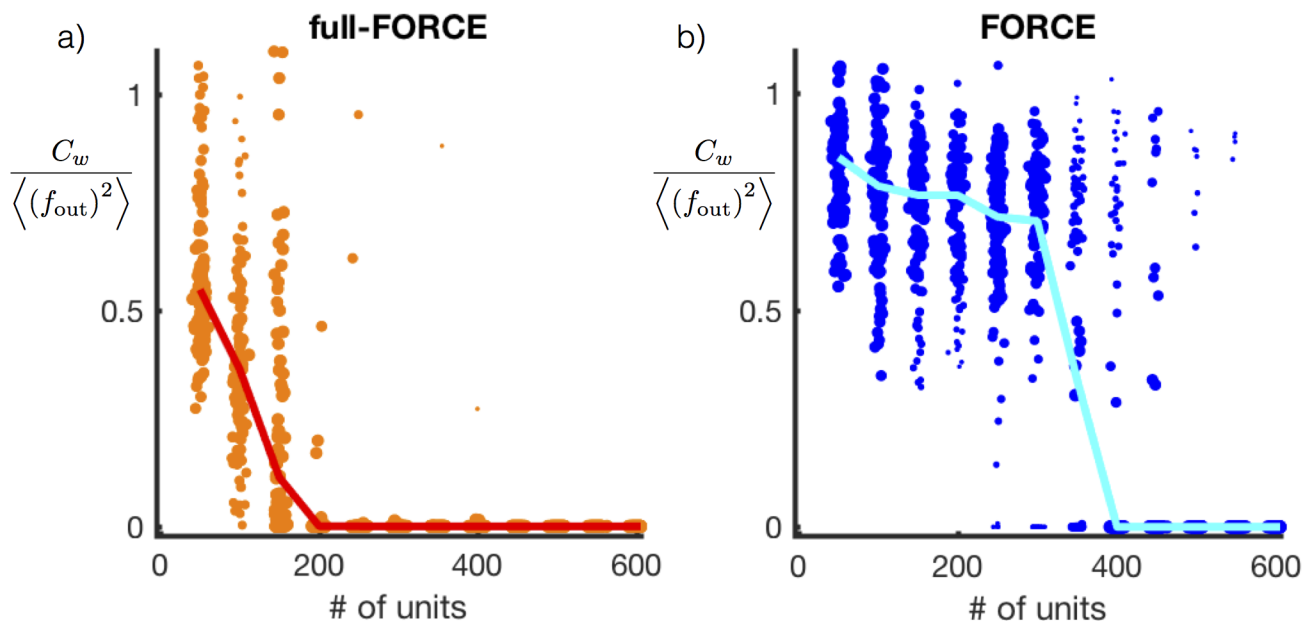
Fig 3. Test error as a function of number of units. Normalized test error following network training for full-FORCE (a) and FORCE (b) as a function of network size. Each dot represents the test error for one random initialization of $\mathbf{J}^D$. Test error was computed for 100 random initializations of $\mathbf{J}^D$ for each value of N. The line indicates the median value across all simulations, and the size of each dot is proportional to the difference of that point from the median value for the specified network size.
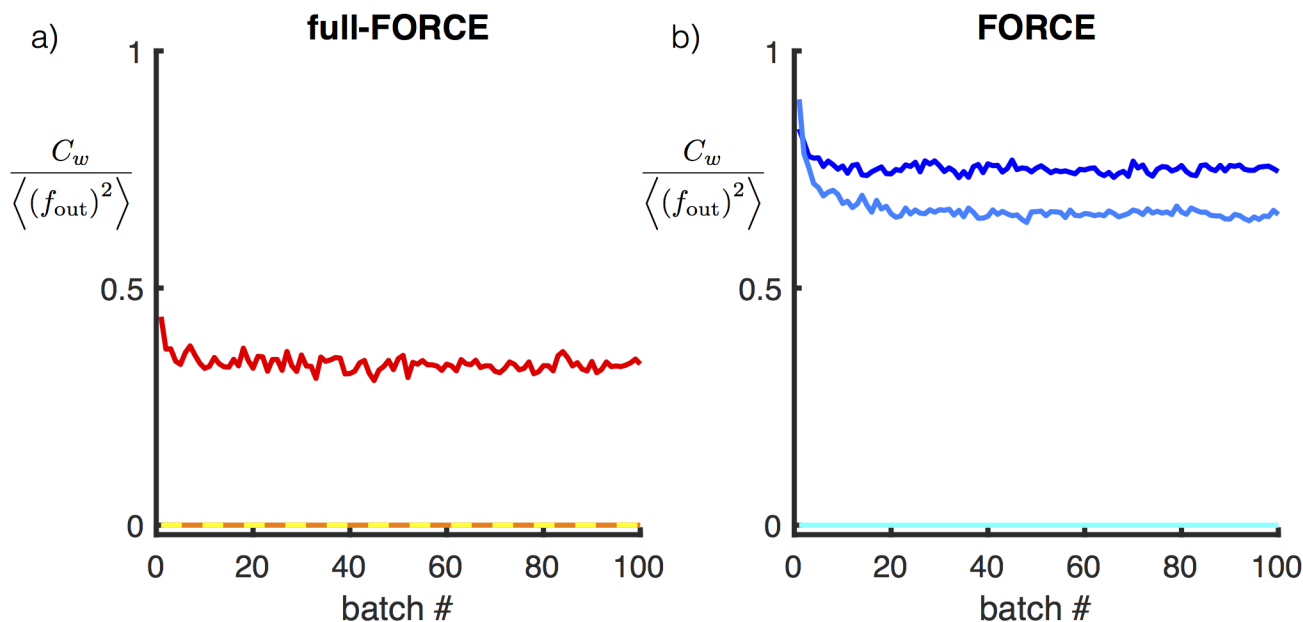
Fig 4. Testing does not improve with more training. Median test error for full-FORCE (a) and FORCE (b) computed across 200 random initializations of $\mathbf{J}^D$ for networks trained on the oscillation task. Three different size networks are shown, 100, 200 and 400 units, where larger networks correspond to lighter colors. The horizontal axis shows the number of batches used to train the network, where each batch corresponds to 100 oscillation periods.
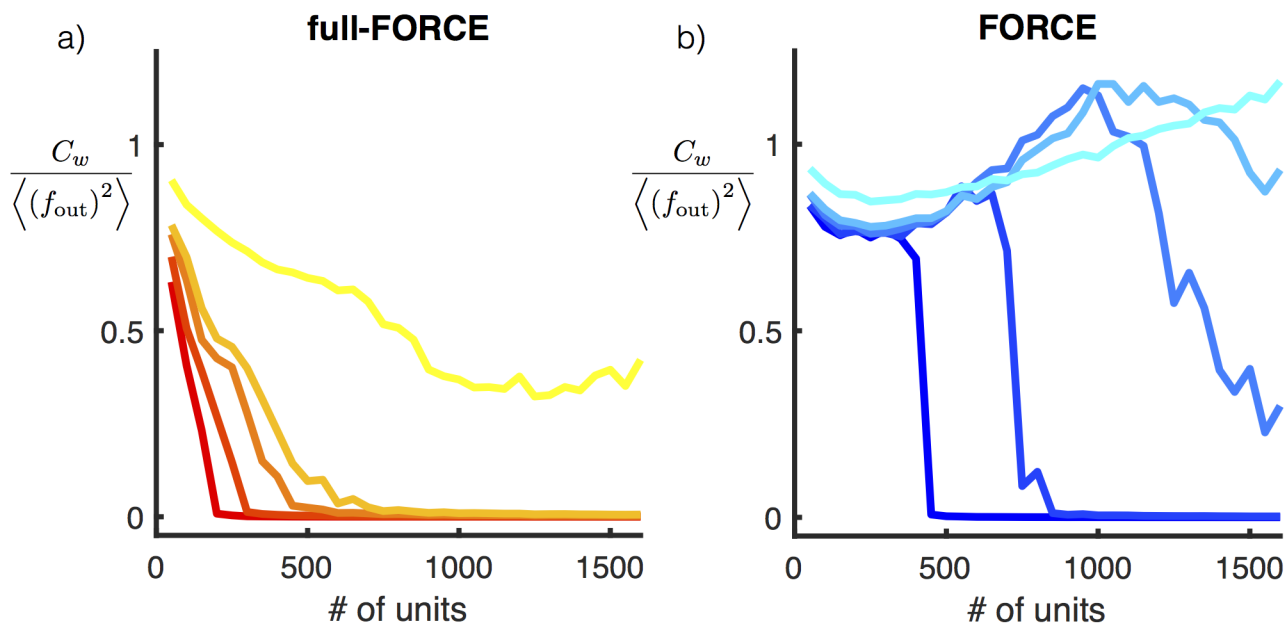
Fig 5. Noise robustness. Median test error for full-FORCE (a) and FORCE (b) computed across 200 random draws of $\mathbf{J}^D$ for various white-noise levels. Increasing noise amplitude corresponds to lighter colors. Levels of noise were determined by $\log(2D)$ = -3, -2, -1, and 0, with D in units of ms^{-1}.

makes the network more sensitive to noise. This effect goes away once learning successfully produces the correct output. This does not occur for full-FORCE, presumably because of its superior stability properties.

Eigenvalues of the recurrent connectivity after learning. To understand the source of the improved performance of full-FORCE compared to FORCE learning, we examined the eigenvalues of the recurrent connectivity matrix **J** after training on the oscillations task and compared them with the eigenvalues of both $\mathbf{J}^D$ and $\mathbf{J}^D + \mathbf{uw}^T$, the connectivity relevant for the FORCE-trained network (Fig 6). The results are for a FORCE and full-FORCE trained network of 300 units that were both successfully trained to perform the oscillations task (for the FORCE network, we used one of the rare cases from Fig 3 where FORCE was successful in training the task).

Mean test error was 1.2×10^{-4} and 1.0×10^{-4} for FORCE and full-FORCE respectively. The eigenvalues of the FORCE trained network are only modestly perturbed relative to the eigenvalue distribution of the initial connectivity $\mathbf{J}^D$, for which the eigenvalues lie approximately within a circle of radius g = 1.5 [22]. The eigenvalues of the full-FORCE network are more significantly changed by learning. Most noticeably, the range of most of the eigenvalues in the complex plane has been shrunk from the circle of radius 1.5 to a smaller area (Fig 6a). Equivalently, the norm of most of the eigenvalues is smaller than for the FORCE-trained network (Fig 6c). The pulling back of eigenvalues that originally had real parts greater than 1 toward real parts closer to 0 reflects the stabilization feature of full-FORCE mentioned previously. By modifying the full connectivity matrix, full-FORCE allows us to take advantage of the expressivity of a strongly recurrent network while culling back unused and potentially harmful modes.

Using additional input to the target-generating networks as hints

The target-generating network of the full-FORCE algorithm uses the "answer" for the task, f_{out}, to suggest a target pattern of activity for the task-performing network. In the procedure
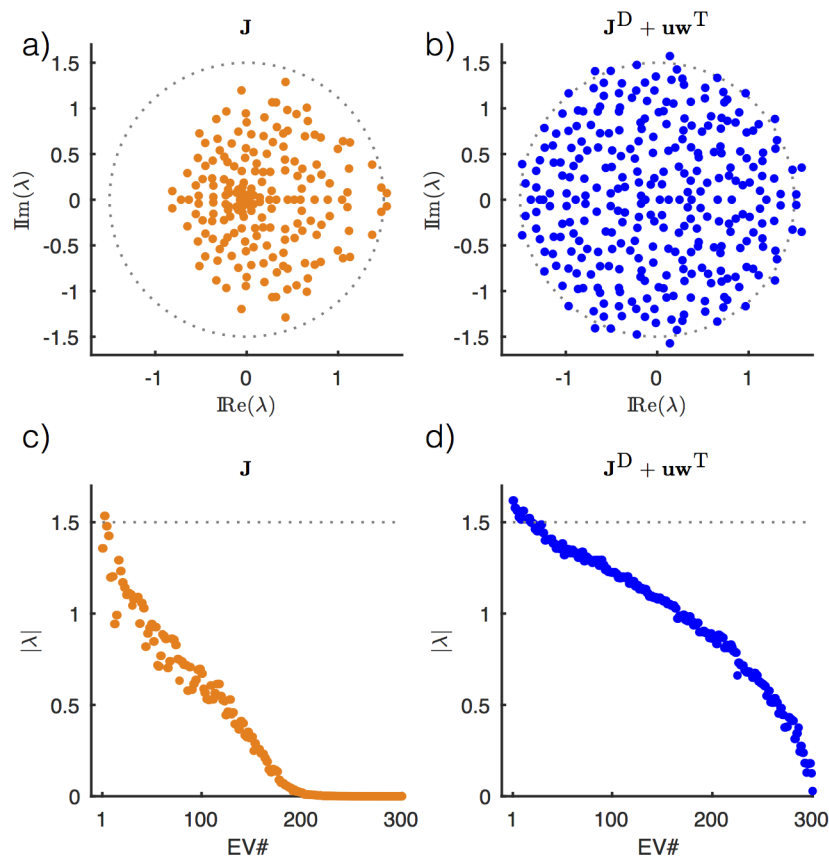
Fig 6. Eigenvalue spectra. (a & b) Eigenvalues of learned connectivity $\mathbf{J}$ and $\mathbf{J}^D + \mathbf{u}\mathbf{w}^T$ for full-FORCE (a) and FORCE (b) respectively. (c & d) Complex norm of eigenvalues for full-FORCE (c) and FORCE (d) respectively. The dotted circle in a and b and dotted line in c and d shows the range of eigenvalues for a large random matrix constructed in the same way as $\mathbf{J}^D$.

we have described thus far, f_{out} is the only supervisory information provided to the target-generating network. This raises the question: why not provide more? In other words, if we are aware of other dynamic features that would help to solve the task or that would lead to a particular type of network solution that we might prefer, why not provide this information to the target-generator and let it be passed on to the task-performing network during training. This extra information takes the form of "hints" [19, 20].

Information about the desired output is provided as an input to the target-generating network, so it makes sense to incorporate a hint in the same way. In other words, we add an extra input term to Eq 7 describing the target-generating network, so now

$$\tau \frac{d\mathbf{x}^D}{dt} = -\mathbf{x}^D + \mathbf{J}^D H(\mathbf{x}^D) + \mathbf{u}f_{\text{out}}(t) + \mathbf{u}_{\text{in}}f_{\text{in}}(t) + \mathbf{u}_{\text{hint}}f_{\text{hint}}(t). \tag{13}$$

The weight vector $\mathbf{u}_{\text{hint}}$ is drawn randomly in the same way as $\mathbf{u}$ and $\mathbf{u}_{\text{in}}$. The key to the success of this scheme lies in choosing the function f_{hint} appropriately. There is no fixed procedure for this, it requires insight into the dynamics needed to solve the task. If the task requires integrating the input, for example, f_{hint} might be the needed integral. If the task requires timing, as in one of the examples to follow, f_{hint} might be a ramping signal. The point is that whatever f_{hint} is, this signal will be internally generated by the task-performing network after successful
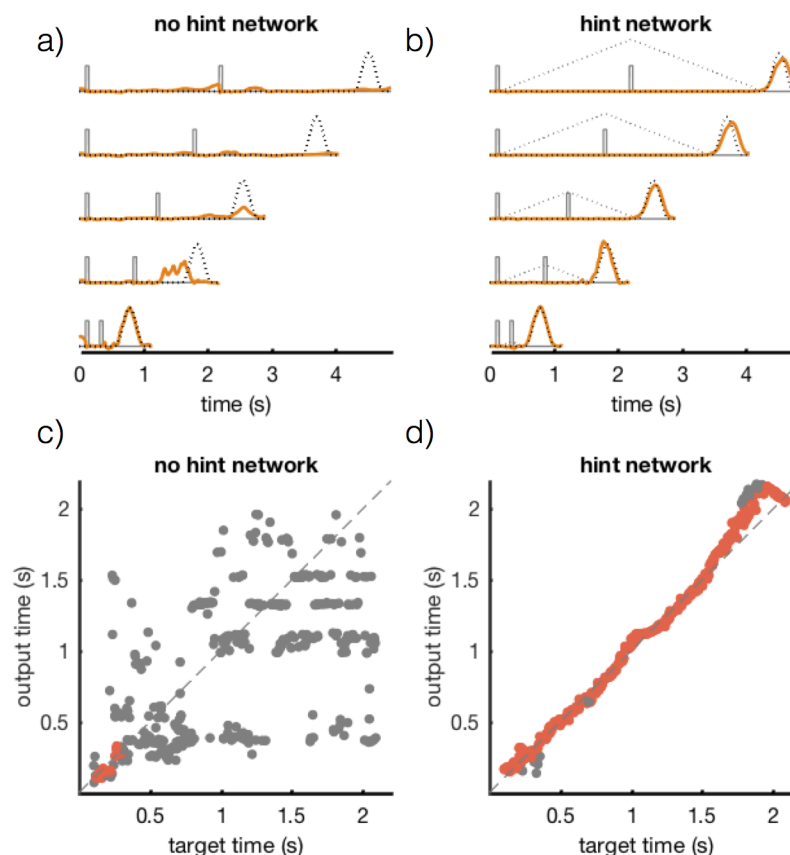
Fig 7. Performance results for networks trained on the interval timing task. (a & b) $f_{in}(t)$ (grey), $f_{hint}(t)$ (grey dotted), $f_{out}(t)$ (black dotted) and $z(t)$ (orange) for a network of 1000 units. Networks trained with full-FORCE learning without (a) and with (b) a hint for various interpulse intervals (100, 600, 1100, 1600 and 2100 ms from bottom to top). (c & d) Target response time plotted against the generated response time without (c) and with (d) hints. Each dot represents the timing of the peak of the network output response on a single test trial. Grey dots indicate that the network output did not meet the conditions to be considered a "correct" trial (see main text). Red dots show correct trials.

https://doi.org/10.1371/journal.pone.0191527.g007

training, and it can then be used to help solve the task. The cost functions used during training with hints are Eq 3 and

$$C_{\mathbf{J}}^{\mathrm{fFhint}} = \left\langle \left| \mathbf{J}H(\mathbf{x}(t)) - \mathbf{J}^{\mathrm{D}}H(\mathbf{x}^{\mathrm{D}}(t)) - \mathbf{u}f_{\mathrm{out}}(t) - \mathbf{u}_{\mathrm{hint}}f_{\mathrm{hint}}(t) \right|^{2} \right\rangle . \tag{14}$$

To illustrate the utility of task hints, we present examples of networks inspired by two standard tasks used in systems neuroscience: interval matching and delayed comparison. In both cases, including task hints drastically improves the performance of the trained network. We also show that using hints allows trained networks to extrapolate solutions of a task beyond the domain on which they were trained. It is worth emphasizing that the hint input is only fed to the target-generating network and only used during training. The hint is not provided to the task-performing network at any time.

Interval matching task. As our first example, we present a network performing an interval matching task (Fig 7) inspired by the "ready-set-go" task investigated by Jazayeri & Shadlen [23]. In this task, the network receives a pair of input pulses $f_{in}(t)$ of amplitude 1.0 and duration 50 ms. The job of the network is to generate an output response at a delay after the second

pulse equal to the interval between the first and second pulses. The time interval separating the two input pulses is selected at random on each trial from a uniform distribution between 0.1 and 2.1 seconds. Following the input pulses, the network must respond with an output pulse $f_{\text{out}}(t)$ that is a smooth, bump-like function (generated from a beta distribution with parameters $\alpha = \beta = 4$) of duration 500 ms and peak amplitude 1.5. Trials begin at random times, with an inter-trial interval drawn from an exponential distribution with a mean of 2.4 seconds. This task is difficult for any form on recurrent network learning because of the long (in terms of the 10 ms network time constant) waiting period during which no external signal is received by the network.

To assess the performance of a trained network, we classified test trials as "correct" if the network output matched the target output with a normalized error less than 0.25 within a window of 250 ms around the correct output time. A network of 1000 neurons, trained by full-FORCE without using a hint performs this task poorly, only correctly responding on 4% of test trials (Fig 7a). The network responded correctly only on those trials for which the interpulse interval was short, less than 300 ms (Fig 7c). This is because, when the desired output pulse $f_{\text{out}}(t)$ is the only input provided, the dynamics of the target-generating network during the waiting period for long intervals are insufficient to maintain timing information. Therefore, we added a hint input to the target-generating network to make its dynamics more appropriate for solving the task. Specifically, we chose f_{hint} to be a function that ramped linearly in time between the first and second pulses and then decreased linearly at the same rate following the second pulse (Fig 7b, grey dotted line). The rate of increase is the same for all trials. A temporal signal of this form should be useful because it ensures that the task-performing network encodes both the elapsed time during the delay and the time when the waiting period is over (indicated by its second zero crossing).

Providing a hint improves performance significantly. An identical network trained with full-FORCE when this hint is included correctly responded to 91% of test trials (Fig 7b). The bulk of the errors made by the network trained with hints were modest timing errors, predominantly occurring on trials with very long delays, as can be seen in Fig 7d. These results illustrate the power of using task hints in our network training framework: including a hint allowed a network to transition from performing the task extremely poorly to performing almost perfectly.

Delayed comparison task. In the previous example, we illustrated the utility of hints on a task that could not be trained without using one. However, not all tasks require hints to be performed successfully. Is there any benefit to include hints for tasks that can be performed without them? Here we train networks on a task that can be performed reasonable well without hints and show that only a network trained with a hint can extrapolate beyond the domain of the training data. Extrapolation is an important feature because it indicates that a more general, and therefore potentially more robust, task solution has been constructed.

We present a version of a delayed comparison task that has been studied extensively by both the experimental and theoretical neuroscience communities [24–26]. In this task, a network receives a pair of input pulses $f_{\text{in}}(t)$ of duration 50 ms. Each pulse has an amplitude that varies randomly from trial to trial, selected from a uniform distribution between 0.125 and 1.875. Furthermore, during training the time interval separating the two input pulses is selected at random from a uniform distribution between 0.1 and 1 second, thereby preventing the network from being able to anticipate the timing of the second pulse. Following the pulses, the network must respond with an output pulse $f_{\text{out}}(t)$ defined as for the interval timing task, but positive if the difference in magnitude of the two input pulses is positive (" + " trials) and negative if the difference was negative ("-" trials) (Fig 8a & 8b). Trials on which the input pulse height difference is small are clearly difficult because smalls errors will cause the network to
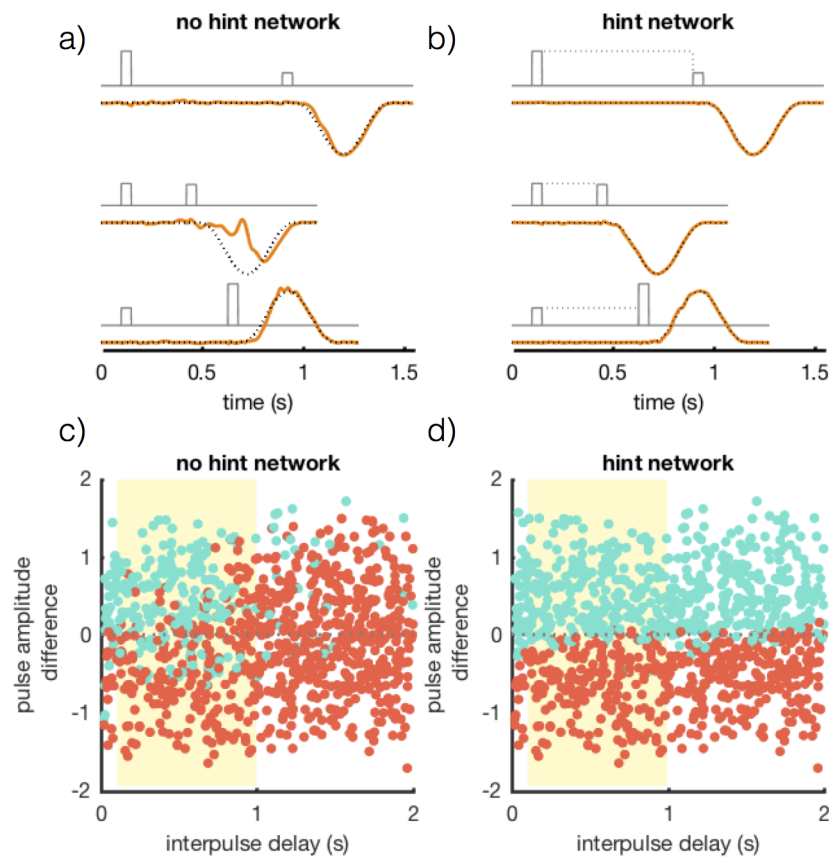
Fig 8. Performance results for networks trained on the delayed comparison task. (a & b) $f_{in}(t)$ (grey), $f_{hint}(t)$ (grey dotted), $f_{out}(t)$ (black dotted) and $z(t)$ (orange) for a network of 1000 units. Networks trained with full-FORCE learning without (a) and with (b) hints. Three different trials are shown from bottom to top: an easy " + " trial, a difficult "-" trial, and an easy "-" trial. (c & d) Test performance for networks trained without (c) and with (d) a hint. Each dot indicates a test trial and the dot color indicates the reported output class (" + " cyan or "-" red). The horizontal axis is the interpulse delay and the yellow region indicates the training domain. The vertical axis indicates the pulse amplitude difference.

https://doi.org/10.1371/journal.pone.0191527.g008

misclassify the input. As in the interval timing task, trials begin at random times with inter-trial intervals of random duration.

To assess the performance of a trained network, we classified test trials as "correct" if the network output matched the target output with a normalized error less than 0.25 and "incorrect" if the network output matched the *opposite* class target output with a normalized error less than 0.25. Trials where the network output did not meet either of these criteria were classified as undetermined and were not included in test performance. Networks trained without a hint correctly classified 71% of test trials. Three example trials are shown in Fig 8a and a summary of test performance can be seen in Fig 8c.

Next we trained an identical network with a hint to see if doing so could increase and extend task performance. The hint signal for this task was a constant input equal to the magnitude of the first pulse that lasted for the entire interpulse interval. A hint of this form has the effect of driving the network dynamics to an input specific fixed-point, thereby constructing a memory signal of the value of the first pulse. Including a task hint increased performance on this task, where the hint trained network could correctly classify 95% of test trials (Fig 8b). More importantly, it allowed the network to extrapolate.

To examine the ability of both trained networks to extrapolate beyond the range of training data, we tested performance with interpulse delays selected uniformly between 20 ms and 2 seconds. Intervals less than 100 ms and greater than 1 second were never seen by the network during training. The network trained without a task hint performed just above chance on these trials, correctly classifying only 53% of trials. The performance of the network trained with the hint, on the other hand, was hardly diminished when compared to its performance over the range for which it was trained; 94% compared to 95% over the training range (Fig 8d). These results illustrate the potential for task hints to simplify the dynamics used by a network to solve a task, thereby allowing a more general solution to emerge, one that supports extrapolation beyond the training data.

Discussion

We have presented a new target-based method for training recurrently connected neural networks. Because this method modifies the full recurrent connectivity, the networks it constructs can perform complex tasks with a small number of units and considerable noise robustness. The speed and simplicity of the recursive least-squares algorithm makes this an attractive approach to network training. In addition, by dividing the problem of target generation and task performance across two separate networks, full-FORCE provides a straightforward way to introduce additional input signals that act as hints during training. Utilizing hints can improve post-training network performance by assuring that the dynamics are appropriate for the task. Like an animal or human subject, a network can fail to perform a task either because it is truly incapable of meeting its demands, or because it fails to "understand" the nature of the task. In the former case, nothing can be done, but in the latter the use of a well-chosen hint during training can improve performance dramatically. Indeed, we have used full-FORCE with hints to train networks to perform challenging tasks that we previously had considered beyond the range of the recurrent networks we use.

The dominant method for training recurrent neural networks in the machine learning community is backpropagation through time [3]. Backpropagation-based learning has been used successfully on a wide range of challenging tasks including language modeling, speech recognition and handwriting generation ([1] and references therein) but these tasks differ from the interval timing and delayed comparison tasks we investigate in that they do not have long periods during which the input to the network is silent. This lack of input requires the network dynamics alone to provide a memory trace for task completion, precisely the function of our hint signal. When using backpropagation through time, the network needs to be unrolled a sufficient number of time steps to capture long time-scale dependencies, which, for our tasks, would mean unrolling the network at least the maximum length of the silent periods, or greater than 2,000 time steps. Given well-known gradient instability problems [27], this is a challenging number of unrollings. While we make no claim that gradient-based learning is unable to learn tasks such as those we study, full-FORCE's success when a well-selected hint is used is striking given the algorithmic simplicity of RLS and architectural simplicity of the units we use. Although we have highlighted the use of hints in the full-FORCE approach, it is worth noting that hints of the same form can be used in other types of learning including traditional FORCE learning and backpropagation-based learning [19, 20]. In these cases, $f_{hint}(t)$ would be added as a second target output, along with $f_{out}(t)$. By requiring the learning procedure to match this extra output, the network dynamics can be shaped appropriately. Within backpropagation-based learning, doing so might avoid gradient instability problems and perhaps release these methods from a reliance on more complex networks units such as LSTMs (Long Short-Term Memory units; [28]).

Studies of recurrent neural networks that perform tasks vary considerably in their degree of "design". At one end of this spectrum are hand-tuned models in which a particular solution is hard-wired into the network by construction (examples of this include [26, 29]). Networks of this type have the advantage of being easy to understand, but their dynamics tends to be simpler than what is seen in the activity of real neurons [24, 30]. At the other extreme are networks that rely on dynamics close to the chaos of a randomly connected network [16], a category that includes the work of Laje & Buonomano [15] and much of the work in FORCE and echo-state networks. This approach is useful because it constructs a solution for a task in a fairly unbiased way, which can lead to new insights. Networks of this type tend to exhibit fairly complex dynamics, an interesting feature [31] but, from a neuroscience perspective, this complexity can exceed that of the data [13, 24]. This has led some researchers to resort to gradient-based methods where regularizers can be included to reduce dynamic complexity to more realistic levels [13]. Another approach is to use experimental data directly in the construction of the model [32, 33]. Regularization and adherence to the data can be achieved in full-FORCE through the use of well-chosen hints imposed on the target-generating network. More generally, hints can be used for a range of purposes, from imposing a design on the trained network to regulating its dynamics. Thus, they supply a method for controlling where a model lies on the designed-to-random spectrum.

By splitting the requirements of target generation and task performance across two networks, full-FORCE introduces a freedom that we have not fully exploited. The dimension of the dynamics of a typical recurrent network is considerably less than its size N [21]. This implies that the dynamics of the target-generating network can be described by a smaller number of factors such as principle components, and these factors, rather than the full activity of the target-generating network, can be used to produce targets, for example by linear combination. Furthermore, in the examples we provided, the target-generating network was similar to the task-performing network in structure, parameters ($\tau = 10$ ms for both) and size. We also made a standard choice for $\mathbf{J}^D$ to construct a randomly connected target-generating network. None of these choices is mandatory, and a more creative approach to the construction of the target-generating network (different τ, different size, modified $\mathbf{J}^D$, etc.) could enhance the properties and performance of networks trained by full-FORCE, as well as extending the range of tasks that they can perform.

Acknowledgments

Author Contributions

Conceptualization: Brian DePasquale, Kanaka Rajan, G. Sean Escola, L. F. Abbott.

Data curation: Brian DePasquale.

Formal analysis: Brian DePasquale, Christopher J. Cueva, G. Sean Escola.

Funding acquisition: L. F. Abbott.

Investigation: Brian DePasquale, Christopher J. Cueva, G. Sean Escola.

Methodology: Brian DePasquale.

Software: Brian DePasquale, Christopher J. Cueva, G. Sean Escola.

Supervision: L. F. Abbott.

Validation: Brian DePasquale.

Visualization: Brian DePasquale.

Writing – original draft: Brian DePasquale, L. F. Abbott.

Writing – review & editing: Brian DePasquale, Christopher J. Cueva, Kanaka Rajan, G. Sean Escola, L. F. Abbott.

References

1. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015; 521: 436–444. https://doi.org/10.1038/nature14539 PMID: 26017442

2. Sussillo D. Neural circuits as computational dynamical systems. Curr. Opin. Neurobiol. 2014; 25: 156–163. https://doi.org/10.1016/j.conb.2014.01.008 PMID: 24509098

3. Werbos PJ. Generalization of backpropagation with application to a recurrent gas market model. Neural Networks. 1988; 1: 339–356. https://doi.org/10.1016/0893-6080(88)90007-X

4. Abbott LF, DePasquale B, Memmesheimer R-M. Building functional networks of spiking model neurons. Nature Neurosci. 2016; 19: 350–355. https://doi.org/10.1038/nn.4241 PMID: 26906501

5. Haykin S. Adaptive Filter Theory. Upper Saddle River, NJ: Prentice Hall; 2002.

6. Lee D-H, Zhang S, Fischer A, Bengio Y. Difference target propagation; 2015. Preprint. Available from: arXiv:1412.7525v5. Cited 25 Nov 2015.

7. Buonomano DV, Merzenich MM. Temporal information transformed into a spatial code by a neural network with realistic properties. Science. 1995; 267: 1028–1030. https://doi.org/10.1126/science.7863330 PMID: 7863330

8. Jaeger H. Adaptive nonlinear system identification with echo state networks. In: Becker S, Thrun S, Obermayer K, editors. Advances in Neural Information Processing Systems 15. Cambridge, MA: MIT Press; 2003. pp. 609–616.

9. Maass W, Natschläger T, Markram H. Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Computation. 2002; 14: 2531–2560. https://doi.org/10.1162/089976602760407955 PMID: 12433288

10. Jaeger H, Haas H. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science. 2004; 304: 78–80. https://doi.org/10.1126/science.1091277 PMID: 15064413

11. Sussillo D, Abbott LF. Generating coherent patterns of activity from chaotic neural networks. Neuron. 2009; 63(4): 544–557. https://doi.org/10.1016/j.neuron.2009.07.018 PMID: 19709635

12. Triefenbach F, Jalalvand A, Schrauwen B, Martens J. Phoneme recognition with large hierarchical reservoirs. Advances in neural information processing systems. 2010; 23:2307–2315.

13. Sussillo D, Churchland M, Kaufman M, Shenoy K. A neural network that finds a naturalistic solution for the production of muscle activity. Nature neurosci. 2015; 18(7): 1025–1033. https://doi.org/10.1038/nn.4042 PMID: 26075643

14. Sussillo D, Abbott LF. Transferring learning from external to internal weights in echo-state networks with sparse connectivity. PLoS One. 2012; 7: e37372. https://doi.org/10.1371/journal.pone.0037372 PMID: 22655041

15. Laje R, Buonomano DV. Robust timing and motor patterns by taming chaos in recurrent neural networks. Nat. Neurosci. 2013; 16: 925–933. https://doi.org/10.1038/nn.3405 PMID: 23708144

16. Sompolinsky H, Crisanti A, Sommers HJ. Chaos in random neural networks. Phys. Rev. Lett. 1988; 61: 259–262. https://doi.org/10.1103/PhysRevLett.61.259 PMID: 10039285

17. Vincent-Lamarre P, Lajoie G, Thivierge J-P. Driving reservoir models with oscillations: a solution to the extreme structural sensitivity of chaotic networks. J. Comput. Neurosci. 2016; https://doi.org/10.1007/s10827-016-0619-3 PMID: 27585661

18. Rajan K, Abbott LF, Sompolinsky H. Stimulus-dependent suppression of chaos in recurrent neural networks. Phys. Rev. E. 2010; 82: 011903. https://doi.org/10.1103/PhysRevE.82.011903

19. Abu-Mostafa YS. Learning from hints. Journal of Complexity. 1994; 10: 165–178. https://doi.org/10.1006/jcom.1994.1007

20. Abu-Mostafa YS. Hints. Neural Computation. 1995; 7: 639–671. https://doi.org/10.1162/neco.1995.7.4.639 PMID: 7584883

21. Rajan K, Abbott LF, Sompolinsky H. Inferring Stimulus Selectivity from the Spatial Structure of Neural Network Dynamics. In: Lafferty J., Williams C. K. I., Shawe-Taylor J., Zemel R.S. and Culotta A. editors. Advances in Neural Information Processing Systems 23. 2010.

22. Girko VL. Circular law. Theory Probab. Appl. 1984; 29: 694–706. https://doi.org/10.1137/1129095

23. Jazayeri M, Shadlen MN. Temporal context calibrates interval timing. Nat. Neurosci. 2010; 13: 1020–1026. https://doi.org/10.1038/nn.2590 PMID: 20581842

24. Barak O, Sussillo D, Romo R, Tsodyks M, Abbott LF. From fixed points to chaos: three models of delayed discrimination. Prog. in Neurobiology. 2013; 103: 214–222.

25. Romo R, Brody CD, Hernandez A, Lemus L. Neuronal correlates of parametric working memory in the prefrontal cortex. Nature. 1999; 399: 470–473.

26. Machens CK, Romo R, Brody CD. Flexible control of mutual inhibition: a neural model of two-interval discrimination. Science. 2005; 307: 1121–1124.

27. Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks. 1994; 5(2): 157–166. https://doi.org/10.1109/72.279181 PMID: 18267787

28. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation. 1997; 9: 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735 PMID: 9377276

29. Seung HS How the brain keeps the eyes still. Proc. Natl. Acad. Sci. 1996; 93: 13339–13344. https://doi.org/10.1073/pnas.93.23.13339 PMID: 8917592

30. Miller P, Brody CD, Romo R, Wang XJ. A recurrent network model of somatosensory parametric working memory in the prefrontal cortex. Cerebral Cortex. 2003; 13: 1208–1218. https://doi.org/10.1093/cercor/bhg101 PMID: 14576212

31. Sussillo D, Barak O. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. Neural Comp. 2013; 25(3): 626–649. https://doi.org/10.1162/NECO_a_00409

32. Fisher D, Olasagasti I, Tank DW, Aksay ERF, Goldman MS. A modeling framework for deriving the structural and functional architecture of a short-term memory microcircuit. Neuron. 2013; 79: 987–1000. https://doi.org/10.1016/j.neuron.2013.06.041 PMID: 24012010

33. Rajan K, Harvey C, Tank D. Recurrent network models of sequence generation and memory. Neuron. 2016; 90: 128–142. https://doi.org/10.1016/j.neuron.2016.02.009 PMID: 26971945