Trajectories in neural space

One of the things we're pretty sure the brain does is generate complex, controllable, time-varying trajectories in neural space. Such trajectories are definitely needed for activating muscles, and they're probably also needed for processing time-varying sensory input. But it's not clear how the brain could generate them. Here we'll discuss one idea.

As usual, we'll consider rate neurons. We'll assume there's an underlying voltage-type variable, which we'll call x_i for neuron i, and that the firing rate is some function of the voltage. Because it's the firing rate that drives voltage, this suggests the following equations,

$$\tau \frac{dx_i}{dt} = \sum_j g W_{ij} \phi(x_i) - x_i \,. \tag{1}$$

Here the nonlinearity that maps voltage to firing rate is ϕ. We have included a scaling factor, g, in front of the weights mainly for historical reasons. Although it is kind of useful; as we'll see, we can adjust g to bring us in and out of the chaotic regime.

Equation (1) is just a recurrent network; if we want it to do something interesting we need a readout. We'll assume the readout is linear in the firing rates,

$$z_\mu = \sum_j A_{\mu j} \phi(x_j) \,. \tag{2}$$

Typically, the number of readout units (the dimensionality of $\mathbf{z}$) is much, much smaller than the number of neurons – think on the order of 1 to 5.

The eventual goal is to train the network so that $z_\mu(t)$ is a desired time-varying signal. For instance, we might want it correspond to signals to muscles that result in some desired limb trajectory. There is, though, clearly a problem: we do not have any way to control the network. To remedy that, we do two things. The first is to add a control signal; the second is to feed the activity, z_μ, back to the network. This results in a set of equations of the form

$$\tau \frac{dx_i}{dt} = \sum_j g W_{ij} \phi(x_j) + \sum_\mu J_{i\mu} z_\mu + \sum_\mu C_{i\mu} u_\mu(t) - x_i. \tag{3}$$

We can combine this with Eq. (2) to eliminate z_μ,

$$\tau \frac{dx_i}{dt} = \sum_j g W_{ij} \phi(x_i) + \sum_{\mu,j} J_{i\mu} A_{\mu j} \phi(x_j) + \sum_\mu C_{i\mu} u_\mu(t) - x_i. \tag{4}$$

This emphasizes the fact that the feedback introduces low rank structure to the connectivity matrix. This may be fundamental to how the brain works, but nobody really knows.

Networks of this type go back more than a decade; see Maass, Matschlager and Markram, *Neural Computation* **14**:2531-2560, 2002, and Jaeger and Haas, *Science* **304**:78-80, 2004. There's a pretty good summary of the history in David Susillo and Larry Abbott's paper (*Neuron* **63**:544-557, 2009), which I'll discuss below. Here I'll give a *very* brief summary.

When networks of the type given in Eq. (4) were invented, the conventional wisdom was that the recurrent connectivity would lead to a rich set of temporal trajectories in the time

domain; those trajectories could then act as temporal basis functions, and a suitable linear combination of them could produce just about any time varying function you want. One way to get a rich set of trajectories is to work in the chaotic regime. The problem, though, is that if the network is chaotic, it's hard to control the trajectories – even with feedback and a control signal. The solution was to work on the edge of chaos: chaotic enough to generate rich trajectories, but not so chaotic that the network is not controllable.

That's where the parameter g comes in. If we ignore the control signal and the feedback from the output units, and we linearize the dynamics around a fixed point, denoted x_{0i}, by letting $x_i = x_{0i} + \delta x_i$), we arrive at the linear equation

$$\frac{d\delta x_i}{dt} = g \sum_j W_{ij} \phi'(x_{0j}) \delta x_j - \delta x_i. \tag{5}$$

When g is very small, all the eigenvalues are close to -1. However, if the first term has at least one positive eigenvalue (the typical case) when g gets large enough one of those eigenvalues will exceed 1, and the fixed point will becomes unstable. This generally leads to chaotic dynamics, and rich temporal trajectories.

In the original work in this field, people used $\phi(x) = \tanh(x)$. In that case, the fixed point is at $x_{0i} = 0$. More importantly, Sompolinsky, Crisanti and Sommers (*Phys. Rev. Lett.* **61**:259262,1988) showed that if the elements of the weight matrix are drawn *iid* from a distribution with variance $1/N$, then the fixed point becomes unstable, and chaotic, when g exceeds 1. However, it's not necessary to work with tanh; any nonlinearity will do.

Networks that admit time-varying dynamics are hard to analyze. However, we can make progress if we set g to zero; that is, if we eliminate the recurrent connectivity. (When we say "recurrent connectivity," we mean W_{ij}. The other recurrent-looking term in Eq. (4), $J_{i\mu} A_{\mu j}$, is – at least in this formulation – due to activity outside the network; see Eqs. (2) and (3).) In that case, Eq. (3) becomes

$$\mathcal{L} x_i = \sum_\mu J_{i\mu} z_\mu + \sum_\mu C_{i\mu} u_\mu(t) \tag{6}$$

where $\mathcal{L}$ is the linear operator

$$\mathcal{L} \equiv \tau \frac{d}{dt} + 1. \tag{7}$$

As is not hard to show, for any suitably well-behaved function $f(t)$

$$\mathcal{L}^{-1} f(t) = \int_{-\infty}^t \frac{dt}{\tau} e^{-(t-t')} f(t'). \tag{8}$$

(You should verify that $\mathcal{L}\mathcal{L}^{-1} f(t) = \mathcal{L}^{-1}\mathcal{L} f(t) = f(t)$.) Thus, we can solve Eq. (6) for x_i, yielding

$$x_i = \mathcal{L}^{-1} \left[\sum_\mu J_{i\mu} z_\mu + \sum_\mu C_{i\mu} u_\mu(t) \right]. \tag{9}$$

Inserting Eq. (9) into (2), and rearranging terms slightly, gives us

$$z_\mu = \sum_i A_{\mu i} \phi \left(\sum_\nu J_{i\nu} \mathcal{L}^{-1} z_\nu + \sum_\nu C_{i\nu} \mathcal{L}^{-1} u_\nu(t) \right) . \tag{10}$$

Finally, defining $\tilde{z}_\mu$ and $\tilde{u}_\mu$ via

$$z_\mu = \mathcal{L}\tilde{z}_\mu = \tau \frac{d\tilde{z}_\mu}{dt} + \tilde{z}_\mu \tag{11a}$$

$$u_\mu = \mathcal{L}\tilde{u}_\mu = \tau \frac{d\tilde{u}_\mu}{dt} + \tilde{u}_\mu, \tag{11b}$$

and using the fact that $\mathcal{L}\mathcal{L}^{-1}$ is the identity, we arrive at

$$\tau \frac{d\tilde{z}_\mu}{dt} = \sum_i A_{\mu i} \phi \left(\sum_\nu J_{i\nu} \tilde{z}_\nu + \sum_\nu C_{i\nu} \tilde{u}_\nu(t) \right) - \tilde{z}_\mu \, . \tag{12}$$

The nonlinear term on the right hand side of this expression is a function purely of $\tilde{z}_\mu$ and $\tilde{u}_\mu$. That function is given by a neural network with one hidden layer. Thus, so long as ϕ is nonlinear and there are enough units, we can implement any function on the right hand side of Eq. (12). Thus, a neural network with low rank structure associated with a loop through another network, but no intrinsic recurrent connectivity, can mimic an arbitrary dynamical system. With this view, the recurrent connectivity – the connectivity that was supposed to be useful for generating rich trajectories – is a nuisance.

So is there a regime in which the recurrent connectivity is useful? There is at least one. Suppose $\mathbf{z}$ is one dimensional. And let's say the control signal is transient, so all it does is set initial conditions. In that case, Eq. (12) becomes (after the control signal is gone)

$$\tau \frac{d\tilde{z}}{dt} = F(\tilde{z}) - \tilde{z} \tag{13}$$

where F is the function implemented by the neural network. Because we're in one dimension, and the dynamics is bounded (since we're modeling a physical system), $\tilde{z}$ must go to a fixed point. In particular, there can't be any periodic trajectories – trajectories that are thought to be important for repetitive motion like walking, breathing, eating, etc.

If, on the other hand, the recurrent connections are strong enough, it is possible to generate periodic trajectories even when $\mathbf{z}$ is one dimensional. The resulting dynamics is, of course, hard to analyze; what people do is simply train the networks. It is not known at this time if the brain makes use of the recurrent connectivity, or if the low rank structure is responsible for the interesting dynamics and the recurrent connectivity is a nuisance.

Learning in recurrent networks

To generate "interesting" dynamics, networks of the type given in Eq. (3) need to be trained. Typically they're trained to generate a target function, which we'll call $z^*_\mu(t)$. That's done by choosing the connection strengths to minimize the error function

$$E \equiv \frac{1}{2} \int dt \sum_\mu \left(z^*_\mu(t) - z_\mu(t) \right)^2 . \tag{14}$$

Using Eq. (2), the error function can be written

$$E \equiv \frac{1}{2}\int dt \sum_\mu \left(z^*_\mu(t) - \sum_i A_{\mu i} r_i(t) \right)^2 . \tag{15}$$

where r_i is the firing rate,

$$r_i(s) \equiv \phi\big(x_i(x)\big). \tag{16}$$

One can minimize error with respect to the recurrent weights, $\mathbf{W}$, the output weights, $\mathbf{A}$, and the feedback weights, $\mathbf{J}$. Here we'll focus on the output weights, $\mathbf{A}$ – mainly because the energy depends very directly on them.

As is typical of learning rules, we use gradient descent,

$$\Delta A_{\mu j} \propto -\frac{\partial E}{\partial A_{\mu j}} . \tag{17}$$

The problem, of course, is that we're dealing with a recurrent network, and so changing the weight at time t has unintended consequences at future times. We could try a greedy algorithm, and adjust the weights according to

$$\Delta A_{\mu j}(t) \propto -\frac{1}{2}\frac{\partial \left(z^*_\mu(t) - \sum_i A_{\mu i} r_i(t) \right)^2}{\partial A_{\mu j}} , \tag{18}$$

which gives us the update rule

$$\Delta A_{\mu j}(t) \propto \big(z^*_\mu(t) - z_\mu(t) \big) r_i(t). \tag{19}$$

This makes sense: if z_μ is too small we want to increase the weight (assuming $\phi(x_i)$ is positive), and if it's too big we want to decrease the weight. It also usually works. But it's very slow – especially if the initial weights are far from their optimal values, and the trajectories are far from what they would be once the weights are learned. To help with learning, training is often done with z_μ in Eq (3) replaced by z^*_μ; that at least ensures that the network is getting the correct input. This was the approach taken by Jaeger and Haas (*Science* **304**:78-80, 2004), and it does speed up training. However, after training, when z_μ is no longer replaced by z^*_μ, the network often doesn't work properly. This isn't especially surprising: we're dealing with a complex dynamical system, and changing the equations (swapping z_μ for z^*_μ) could easily change the stability of the desired trajectory.

Until 2009, most neuroscientists paid very little attention to this problem. But then David Susillo and Larry Abbott introduced a learning rule, called FORCE learning, that converged very fast (*Neuron* **63**:544-557, 2009). (FORCE stands for first-order reduced and controlled error, but nobody I know can remember that.) Since then there has been an explosion of interest among neuroscientists. It's not exactly clear why the sudden explosion; as we'll see, FORCE learning is clearly not biologically plausible, so the learning rule has nothing to do with the brain. And the networks Susillo and Abbott used were pretty standard. It just goes to show: neuroscience is as much about fashion as anything else. That said, I view

the interest in these kinds of networks as a good thing, since it's critically important to understand networks that generate nontrivial time-varying activity.

So what is FORCE learning? It's actually just the recursive least-squares algorithm, and works as follows. Without loss of generality, we can focus on one value of μ. Let's also turn the integral in Eq. (15) into a sum, and add a regularization,

$$E_\mu(t) = \frac{1}{2}\sum_{s=0}^{t}\left(z_\mu^*(s) - \sum_i A_{\mu i} r_i(s)\right)^2 + \frac{\alpha}{2}\sum_i A_{\mu i}^2. \tag{20}$$

Minimizing $E_\mu(t)$ with respect to $A_{\mu i}$ yields the usual least-squares solution,

$$A_{\mu i}(t) = \sum_j P_{ij}(t) h_{\mu j}(t) \tag{21}$$

where $P_{ij}(t)$ is the inverse of the second moment of the activity (plus a regularization term),

$$P_{ij}^{-1}(t) \equiv \alpha\delta_{ij} + \sum_{s=0}^{t} r_i(s) r_j(s) \tag{22}$$

where δ_{ij} is the Kronecker delta (it's 1 if $i = j$ and 0 otherwise) and $h_{\mu i}(t)$ is the proportional to the correlation between the activity and z_μ^*,

$$h_{\mu i}(t) \equiv \sum_{s=0}^{t} r_i(s) z_\mu^*(s)\,. \tag{23}$$

To implement these update rules we have to invert a matrix, which doesn't seem easy – especially for large networks. However, when done online, that inversion turns out to be reasonably straightforward, leading also to reasonably straightforward update rules for the weights, $A_{\mu i}$. To derive those rules, it is convenient to switch to vector/matrix notation, for which Eq. (21) becomes

$$\mathbf{A}_\mu(t) = \mathbf{P}(t)\cdot\mathbf{h}_\mu(t). \tag{24}$$

To update this equation, we need both $\mathbf{P}$ and $\mathbf{h}_\mu$ on the next timestep. To make the equations look less intimidating, we'll use $t+1$ for the next time step, although "1" should really be "Δt".

The update rule for $\mathbf{h}_\mu$ is easy; using Eq. (23), we have

$$\mathbf{h}_\mu(t+1) = \mathbf{h}_\mu(t) + \mathbf{r}(t+1) z_\mu^*(t+1)\,. \tag{25}$$

The update rule for $\mathbf{P}$ is a only slightly harder,

$$\mathbf{P}(t+1) = \left(\alpha\mathbf{I} + \sum_{s=0}^{t+1}\mathbf{r}(s)\mathbf{r}(s)\right)^{-1} = \left(\mathbf{P}(t)^{-1} + \mathbf{r}(t+1)\mathbf{r}(t+1)\right)^{-1} \tag{26}$$

where $\mathbf{I}$ is the identity matrix. It's not hard to compute the inverse on the right hand side, yielding

$$\mathbf{P}(t+1) = \mathbf{P}(t) - \frac{\mathbf{P}(t)\cdot\mathbf{r}(t+1)\,\mathbf{r}(t+1)\cdot\mathbf{P}(t)}{1+\mathbf{r}(t+1)\cdot\mathbf{P}(t)\cdot\mathbf{r}(t+1)}\,. \tag{27}$$

We are now in a position to find $\mathbf{A}_\mu(t+1)$. Using Eq. (24), we have

$$\mathbf{A}_\mu(t+1) = \left(\mathbf{P}(t) - \frac{\mathbf{P}(t)\cdot\mathbf{r}(t+1)\,\mathbf{r}(t+1)\cdot\mathbf{P}(t)}{1+\mathbf{r}(t+1)\cdot\mathbf{P}(t)\cdot\mathbf{r}(t+1)}\right)\cdot\big(\mathbf{h}_\mu(t) + \mathbf{r}(t+1)z^*_\mu(t+1)\big)\,. \tag{28}$$

Expanding the terms and performing a small amount of algebra gives us

$$\mathbf{A}_\mu(t+1) = \mathbf{P}(t)\cdot\mathbf{h}_\mu(t) + \frac{\mathbf{P}(t)\cdot\mathbf{r}(t+1)\big(z^*_\mu(t+1) - \mathbf{r}(t+1)\cdot\mathbf{P}(t)\cdot\mathbf{h}_\mu(t)\big)}{1+\mathbf{r}(t+1)\cdot\mathbf{P}(t)\cdot\mathbf{r}(t+1)}\,. \tag{29}$$

As we can see from Eq. (24), the first term on the right hand side is just $\mathbf{A}_\mu(t)$. For the second term, we note, via Eq. (27), that

$$\mathbf{P}(t+1)\cdot\mathbf{r}(t+1) = \frac{\mathbf{P}(t)\cdot\mathbf{r}(t+1)}{1+\mathbf{r}(t+1)\cdot\mathbf{P}(t)\cdot\mathbf{r}(t+1)}\,. \tag{30}$$

Using these two observations, we arrive at

$$\mathbf{A}_\mu(t+1) = \mathbf{A}_\mu(t) + \mathbf{P}(t+1)\cdot\mathbf{r}(t+1)\big(z^*_\mu(t+1) - \mathbf{r}(t+1)\cdot\mathbf{A}_\mu(t)\big)\,. \tag{31}$$

This update rule for $\mathbf{A}_\mu$, combined with Eq. (27) for $\mathbf{P}$, is the FORCE learning algorithm. This learning rule is non-local (since it keeps track of the full covariance matrix). It is thus not biologically plausible. Nor is there an obvious guarantee that it will converge. That's because when you change $\mathbf{A}_\mu$, you change the dynamics; that's not taken into account when minimizing E_μ. Nevertheless, in practice it works pretty well.