

## FULL SIMULATION DETAILS

### Simulation details for Fig. 1 and Supplementary Fig. 1

We simulate two different electromyograms (EMGs) (see Methods Section 1.4) of muscle activities (initial reach and target reach) that each last 0.5 s (see Figs. 1a,f). We use a network of  $N = 200$  neurons and sample transient neuronal firing rates that last 0.5 s following the initial condition  $\mathbf{x}_0$  of the neuronal activity (see Methods Section 1.1). We fit the readout weights over 100 trials, in which we add white Gaussian noise to the initial condition  $\mathbf{x}_0$  (with a signal-to-noise ratio of 30 dB) using least-squares regression so that the network output, with all gains set to 1, generates the initial reach (see Methods Section 1.5). We use the same readout weights throughout all training, and we use only one readout unit for each simulation.

In Fig. 1c, we plot the dynamics of three example neurons with all gains set to 1 (black) and all gains set to 2 (blue).

For each training iteration of the neuronal gains (to generate a target movement), we use the initial condition  $\mathbf{x}_0$  at time  $t = 0$  (see Methods Section 1.1). We calculate the subsequent network output as described in Methods Section 1.5, and we update the neuronal gains according to Eqn. (8). We repeat this process for 18,000 training iterations (which corresponds to 2.5 hours of training time), which is enough training time for the error to saturate (see Fig. 1d).

We run 10 independent training sessions on the same target, and we plot these results in Figs. 1d,e. For each of the 10 trained gain patterns  $\mathbf{g}$ , we plot the change in the spectral abscissa of  $\mathbf{W} \times \text{diag}(\mathbf{g})$  (i.e., the largest real part in the spectrum of  $\mathbf{W} \times \text{diag}(\mathbf{g})$ ) in Supplementary Fig. 1a. We observe an increase in the spectral abscissa after training. Although this change seems substantial, the resulting firing-rate activity does not change dramatically (see Supplementary Fig. 1b).

Additionally, we generate 100 network outputs for each of the 10 trained gain patterns using 100 different instances of white Gaussian noise added to the initial condition  $\mathbf{x}_0$  with a signal-to-noise ratio of  $s$  dB (where we consider values of  $s$  between 1 and 30 dB in increments of 1). We then calculate the square of the Euclidean 2-norm between each network output and the network output that we obtain when we do not add noise to the initial condition. We call these squared errors  $e_1$ . (This vector has 1,000 entries, with one entry for each network output.) We also generate 1,000 outputs with all gains set to 1 using 1,000 different instances of white Gaussian noise added to the initial condition  $\mathbf{x}_0$  with a signal-to-noise ratio of  $s$  dB. (We again consider values of  $s$  between 1 and 30 dB in increments of 1.) We then calculate the square of the Euclidean 2-norm between each of these network outputs and the network output that we obtain with all gains set to 1 and no noise added to the initial condition. We call these squared errors  $e_2$ . For each signal-to-noise ratio  $s$ , we plot the mean and standard deviation of  $e_1$  (i.e., the squared error corresponding to the trained gain patterns) in red and  $e_2$  (i.e., the squared error corresponding to all gains set to 1) in blue in Supplementary Fig. 1d. We obtain very similar errors for both the trained and untrained (i.e., all gains set to 1) gain patterns, except for large (i.e., approximately larger than 25 dB) signal-to-noise ratios. For the outputs that we show in Fig. 1f, we add white Gaussian noise to the initial condition

$\mathbf{x}_0$  with a signal-to-noise ratio of 30 dB using one of the trained gain patterns and with all gains equal to 1.

To generate the correlation matrices that we show in Supplementary Fig. 1b, we calculate the Pearson correlation coefficient of the neuronal firing rates between all pairs of neurons in the recurrent neuronal network. Therefore, each entry in the matrix indicates the extent to which the neuronal firing rates are similar for a pair of neurons over the duration of the movement (i.e., 0.5 s). In Supplementary Fig. 1b, we show correlation matrices for examples in which all gains are set to 1 and for two example learned gain patterns. We use the same initial condition  $\mathbf{x}_0$  that we used during training.

We also study whether neuronal firing rates correlate more positively with a target movement after training than before training. To quantify the similarity between the neuronal firing rates and the target output, we calculate — for each of the 10 training sessions that we used in Fig. 1d — the Pearson correlation coefficient of the neuronal firing rates between each neuron and the target output. In Supplementary Fig. 1e, we plot the mean Pearson correlation coefficient across all neurons for the case in which all gains are set to 1 (i.e., before training) and for each of the 10 learned gain patterns (i.e., after training). There is a significant (with a p-value of  $p \approx 0.002$ ) change in the mean Pearson correlation coefficient before training versus after training using a paired Wilcoxon signed rank one-sided test. For the gain pattern that produces the largest change in the mean correlation coefficient (we show this with the grey line in the left panel of Supplementary Fig. 1e), we plot the distribution of changes in the correlation coefficients for all neurons in the bottom right panel of Supplementary Fig. 1e. We see that most values are larger than 0, so the neuronal firing rates become more positively correlated with the target output after learning. We also show an example of a substantial change in the neuronal firing rate of one neuron in the top right panel of Supplementary Fig. 1e.

In another computational experiment, we generate 10 different target muscle activities (see Methods Section 1.4) and, independently for each movement, we train either the neuronal gains, the recurrent synaptic weight matrix  $\mathbf{W}$ , the initial condition  $\mathbf{x}_0$ , or a rank-1 perturbation of the recurrent synaptic weight matrix using a gradient-descent training procedure (with gradients that we obtain from back-propagation [1]). Before training, we use the 200-neuron stability-optimised network, initial condition  $\mathbf{x}_0$ , and readout weights that we used in Fig. 1. Specifically, before any training, the network output is the black curve that we show in Fig. 1f. The cost function for the training procedure is the squared error between the network output and the target movement scaled by the total sum of squares of the target movement (i.e., Eqn. (7)). We run the gradient-descent training procedure until the difference between the cost function at successive training iterations is below  $10^{-5}$  (i.e., until the cost saturates to a small value). When we train the recurrent synaptic weight matrix  $\mathbf{W}$ , after each weight update, we set any positive inhibitory weights to zero and we set any negative excitatory weights to zero. For the rank-1 perturbation, we independently train vectors  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{200 \times 1}$  to reduce the error between the network output, which we obtain from the neuronal firing rates in Eqn. (1) with  $\mathbf{W}$  replaced by  $\mathbf{W} + \mathbf{u}\mathbf{v}^\top$ , and the target movement. Before training, the elements of  $\mathbf{u}$  and  $\mathbf{v}$  are chosen from a Gaussian distribution with a mean 0 and standard deviation 0.05. In Supplementary Fig. 1f, we plot the errors for 10 different target movements

for each of our 4 different training approaches.

### Simulation details for Fig. 2

For this figure, we train neuronal gains on the same task as the one that we showed in Fig. 1d — that is, we independently train 10 gain patterns to generate the target output that we showed in orange in Fig. 1f — using 3 alternative models. We use neuron-specific modulation for these simulations. (This contrasts with our group-based gain modulation.) We fit the readout weights so that, prior to any training (i.e., with all gains set to 1), the network output is the same in each model. (See the black curve in Fig. 1f.) We show the mean error during training in Fig. 2a. The red curve is the same error curve that we plotted in Fig. 1d, but we now use a logarithmic vertical-axis scale.

We also train the neuronal gains on the same task as above, but now using a ramping input to the network (to simulate preparatory activity prior to movement onset [2,3]). We use the same ramping input function as the one that was used in Ref. [2]. It is  $\exp(t/\tau_{\text{on}})$  for  $t < 0$  s and  $\exp(-t/\tau_{\text{off}})$  after movement onset ( $t \geq 0$ ), with an onset time of  $\tau_{\text{on}} = 400$  ms and an offset time of  $\tau_{\text{off}} = 2$  ms. Gain changes that result from learning now also affect the neuronal activity at  $t = 0$  (i.e., at movement onset). We again run 10 independent training sessions, and we observe results that are qualitatively similar to those we saw in Fig. 1d. (See the blue curve in Fig. 2a.)

We also train a ‘chaotic’ [4] variant of our model (see Methods Section 1.3, where we describe how we construct such a model), and we train on the same target movement that we mentioned above. We use the first 0.5 s of neuronal activity. We observe a very similar error reduction over training iterations (see the grey curve in Fig. 2a) as we saw in Fig. 1d. (Compare the grey and red curves in Fig. 2a.)

Finally, we use an alternative learning rule (see Eqns. (10) and (11)) in which learning stops automatically when the difference between network outputs over successive training iterations becomes sufficiently small (see Methods Section 1.7). In Fig. 2a, we plot the error reduction using this alternative learning rule in purple. Using this alternative learning rule, we obtain a smaller error for this task (compare the purple and red curves in Fig. 2a), and learning stops after approximately 10,000 training iterations on average.

In Fig. 2b, we plot the firing rates of 4 example neurons for each of these 4 models both before and after training the neuronal gains.

### Simulation details for Fig. 3 and Supplementary Fig. 2

For the same task as in Fig. 1, we plot the results of using random and specialized groupings (see Methods Section 1.9), as well as the neuron-specific result from Fig. 1d, in Supplementary Fig. 2a. We use the same readout weights that we used in Fig. 1.

We now give details for Figs. 3b,c and Supplementary Figs. 2b–d. We generate 5 different target outputs and run 10 independent training sessions for each target. For the random groupings (see Methods Section 1.9), we use different independently-generated random groups for each simulation. For the specialized groups (see Methods Section 1.9), for a given number of groups, we use the same grouping in all simulations. We plot the results of using 10 or 20 groups with either random or specialized groups in Figs. 3b,c and Supplementary Figs. 2b,c.

We now explain how we determine specialized groups that are shared by multiple movements (i.e., we use the same grouping for learning multiple movements); see the plots in Fig. 3c and Supplementary Figs. 2b–d. We apply  $k$ -means clustering (where  $k$  is the desired number of groups) across all of the gain patterns that we obtain using neuron-specific modulation for each of the movements. That is, we apply  $k$ -means clustering to a matrix of size  $N \times 10 \cdot q$ , where  $N$  is the number of neurons and  $q$  is the number of movements (and, equivalently, the number of gain patterns). We also use the specialized grouping that we obtain for 20 groups that is shared across 5 movements (see Supplementary Figs. 2b) to learn 10 hitherto-untrained movements. We plot these results in Supplementary Fig. 2d.

For the task that we just described above, we consider various different numbers of groups (using random groupings) for networks with  $N = 100$ ,  $N = 200$ , and  $N = 400$  neurons. We again perform 10 independent training sessions for each network, target, and number of groups. We fit the readout weights so that each scenario generates the same network output when all gains are set to 1. The readout weights remain fixed throughout training. We plot these results in Fig. 3d and Supplementary Figs. 2e–h.

We now give details for Figs. 3e,f. When we use multiple readout units, we generate 10 different initial and target outputs for each readout unit. For example, for 2 readout units, we generate 10 different initial and target outputs for each of units 1 and 2. We run independent training sessions for these 10 sets of target outputs and calculate mean errors across the 10 training sessions. For a given number of readout units, we use the same sets of initial and target outputs for all 3 network sizes and each number of random modulatory groups. We thus fit readout weights so that each scenario generates the same output with all gains set to 1. The readout weights remain fixed throughout training. We use 60,000 (instead of 18,000) training iterations to ensure error saturation.

### **Simulation details for Figs. 4, 5f, and Supplementary Figs. 3,4**

To create libraries of learned movements, we train a network of 400 neurons and 40 random groups (see Methods Section 1.9) on each of 100 different target movements independently. (In other words, this generates 100 different gain patterns, with one for each movement.) In Supplementary Fig. 3a, we plot the distribution of gains that we obtain after training across all 100 gain patterns. We plot all 100 outputs from these 100 learned gain patterns in Supplementary Fig. 3b. We also generate 100 new gain patterns by sampling uniformly at random from the distribution in Supplementary Fig. 3a and plot the output of each of these gain patterns in Supplementary Fig. 3c. These

outputs are much more homogeneous than the learned gain patterns in Supplementary Fig. 3b, and they likely would not constitute a good basis set for movement generation.

For library sizes of  $l \in \{1, 2, \dots, 50\}$ , we choose 100 samples of  $l$  movements (from the learned gain patterns and their outputs) uniformly at random without replacement for each  $l$ . We then fit the set of movements in each of the 100 sample libraries using least-squares regression for each of 100 hitherto-untrained novel target movements. We constrain the fitting coefficients  $c_j$  from the least-squares regression by requiring that  $c_j \geq 0$  for all  $j$  and  $\sum_{j=1}^l c_j = 1$ . That is, we consider convex combinations of the coefficients  $c_j$ . We calculate the fit error (i.e., the error between the fit and the target), the output error (i.e., the error between the output and the target), and the error between the fit and the output for each of the 100 novel movements, each of the 100 library samples, and each  $l$ .

For each  $l$  and for each randomly-generated combination of library elements (see the paragraph immediately above), we order the 100 novel target movements based on the error between the output and the fit, and we select the one that is the 50th smallest (i.e., close to the median error). We then extract the output and fit errors for this target and repeat this procedure for each of the 100 randomly-generated combinations of library elements and for  $l = 1, \dots, 50$ . We plot these results in Fig. 4c and Supplementary Fig. 3g. In Fig. 4, we plot results for  $l \in \{1, 2, \dots, 20\}$ ; in Supplementary Fig. 3, we plot results for  $l \in \{1, 2, \dots, 50\}$ . Observe that there is only a small change in the errors between  $l = 20$  and  $l = 50$ .

In Fig. 4b, for an example target (and for  $l = 2, l = 4, l = 8$ , and  $l = 16$ ), we plot the output and fit that produce the 50th-smallest error between the output and the target across the 100 randomly-generated libraries. In Supplementary Fig. 3e, we calculate the median error over the 100 target movements and we plot the distribution of these median errors over the 100 randomly-generated combinations of library elements for  $l = 5$  and  $l = 20$ .

Additionally, for each  $l$  and for each of the 100 target movements, we order the 100 combinations of library elements based on the error between the output and the fit, and we select the one that is the 50th smallest. We then extract the output and fit errors for this combination and repeat this procedure for each of the 100 target movements and for  $l = 1, \dots, 50$ . We plot these results in Supplementary Fig. 3h. This indicates that we obtain qualitatively similar results if we average over the 100 target movements or if we instead average over the 100 combinations of library elements. In Fig. 4d and Supplementary Fig. 3d, we first calculate the median error over the 100 target movements for each  $l$  and for each of the 100 combinations of library elements. We then plot the median of these errors over the 100 combinations of library elements for each  $l$ .

We also calculate the Pearson correlation coefficient between the output and the fit errors for each  $l$  when taking the 50th-smallest error across the 100 novel target movements (see Supplementary Fig. 3i) or across the 100 randomly-generated samples (see Supplementary Fig. 3j).

We also repeat these simulations for the baseline rate  $r_0 = 5$  Hz in Eqn. (2). We plot the results of these simulations in Fig. 5f (see the next subsection) and Supplementary Fig. 4, and we note that we obtain very similar results to those that we obtained for  $r_0 = 20$  Hz.

### Simulation details for Figs. 5a–e

We now describe the details of our simulations when using a baseline rate of  $r_0 = 5$  Hz.

For the 200-neuron network that we used in Fig. 1, we plot the (relative to baseline) firing rate  $f(x)$  (see Eqn. (2)) of 20 excitatory and 20 inhibitory neurons in Fig. 5 with (panel (a))  $r_0 = 20$  Hz in Eqn. (2) and (panel (b))  $r_0 = 5$  Hz. In Fig. 5c, we plot the relative firing rate of all neurons over time versus the relative firing rate when using a linear gain function (i.e.,  $f(x_i; g_i) = g_i x_i$ ) for the cases of (black)  $r_0 = 20$  Hz and (blue)  $r_0 = 5$  Hz. We set all of the gains to 1 for these simulations.

We also train a recurrent neuronal network on the same task as the one that we showed in Figs. 1d–f, except with a baseline rate of  $r_0 = 5$  Hz. We plot these results in Fig. 5d–e and compare them to our observations for  $r_0 = 20$  Hz. For the 10 noisy initial conditions that we used to generate the outputs in the inset in Fig. 5d, we add white Gaussian noise to the initial condition  $x_0$  with a signal-to-noise ratio of 30 dB. In other words, we generate noise in the same manner as we did in Fig. 1f.

### Simulation details for Fig. 6 and Supplementary Figs. 5,6,8

We now describe our simulations for learning target activity that lasts longer than 0.5 s. In each of these simulations, we use a network of 400 neurons and 40 random modulatory groups. (See Methods Section 1.9 for details on how we determine such groups.) We construct ‘slow’ (2.5 s) target movements with  $\sigma = 550$  ms and  $\ell = 250$  ms in Eqn. (5). We then construct a ‘fast’ (0.5 s) variant of each movement. Each movement variant has 500 evenly-spaced points (see Methods Section 1.4). We sample the fast variant using 100 evenly-spaced points, and we then augment 400 instances of 0 values to the final 2 s of the movement to ensure that both movement variants have the same length. (See the top right of Fig. 6a.)

**Details for Fig. 6b, Supplementary Figs. 5a,c,e, and Supplementary Fig. 8.** For Fig. 6b, we fit readout weights using least-squares regression, such that with all gains set to 1, the network output approximates the fast variant. We then train gain patterns using our learning rule in Eqns. (8) and (9) so that the network output generates the slow-movement variant. (The initial condition  $x_0$  and readout weights remain fixed.) We use 60,000 training iterations, and we run 10 independent training sessions for each of 10 different target movements. We plot one such movement in Fig. 6b, and we plot results of all simulations in Supplementary Figs. 5a,c. For Supplementary Fig. 8, we perform the same task except that we scale the amplitude of the slow-movement variant by the factor  $1/25$ . Scaling the slow-variant target movement by this factor corresponds to the same actual movement but lasting 5 times longer (see Methods Section 1.4). In Supplementary Fig. 8, we show results for the same example that we plotted in Fig. 6b.

**Details for Fig. 6c and Supplementary Figs. 5b,d,f,g.** We wish to obtain neuronal dynamics that are less sensitive to noisy initial conditions than those that we generated from gain patterns that we obtained from our learning rule (i.e., those that we plot in Supplementary Figs. 5a). For example, in Fig. 6b, the neuronal firing rates have decayed substantially towards baseline after approximately 0.75 s, even though the output activity is close to its maximum value. Therefore, a small change in the initial condition would likely substantially affect the neuronal activity for times after approximately 0.75 s. We therefore perform the task that we described in the paragraph above (i.e., generating a slow-movement variant by changing neuronal gains) using a gradient-descent training procedure with gradients that we obtain from back-propagation [1]. Together with learning the gain pattern for the slow variant, we jointly optimize a single set of readout weights (shared by both the fast-movement and slow-movement variants), as we discussed in Methods Section 1.5, as part of the same training procedure. The gains are still fixed at 1 for the fast variant. The cost function for the training procedure is equal to the squared Euclidean 2-norm between actual network outputs and the corresponding target outputs at both fast and slow speeds plus the Euclidean 2-norm of the readout weights, where the latter acts as a regularizer. We run gradient descent for 500 iterations, which is well after the cost has stopped decreasing.

Using the target movement from Fig. 6b, we plot the output of the back-propagation training procedure in Fig. 6c, and we plot results of all simulations in Supplementary Figs. 5b,d on the same 10 target movements as those that we used in Supplementary Fig. 5a. In Supplementary Fig. 5g, for the outputs in Figs. 6b,c, we add white Gaussian noise with a signal-to-noise ratio of 4 dB to the initial condition. We observe that the outputs from the back-propagation training procedure are less sensitive than the outputs from the learning rule to noisy initial conditions.

**Details for Supplementary Figs. 5h–j.** In these simulations, we train a single gain pattern that is shared by  $m$  different movements, which each last 2.5 s and where each movement corresponds to a different initial condition (IC). To generate a collection of  $m$  such ICs, in which each IC evokes neuronal activity of approximately equal amplitude with all gains set to 1, we randomly rotate the top  $m$  eigenvectors of the observability Gramian of the matrix  $\mathbf{W} - \mathbf{I}$  [2]. Specifically, we do this by creating a matrix of  $m$  columns — one for each of these  $m$  eigenvectors — and right-multiplying this matrix by a random  $m \times m$  orthogonal matrix (which we obtain via a QR decomposition of a random matrix with elements drawn from a normal distribution with mean 1 and standard deviation 1).

Given  $m$  ICs, we uniformly-at-random choose  $m$  fast target movements and their slow counterparts out of a fixed set of 10 different movements. We then train a recurrent neuronal network to generate the correct fast and slow target movements by optimizing a single set of readout weights (shared by both fast and slow variants) and a single gain pattern that generates the slow variants (where we set the gains for each of the fast variants to 1). We train using the same gradient-descent method with back-propagation that we described above for Fig. 6c. We plot the results as a function of the number  $m$  of movement–IC pairs (see Supplementary Figs. 5h,i) for 10 independent draws of the ICs that we just described above.

**Details for Fig. 6d; top panel.** For each of the 10 trained movements in Supplementary Figs. 5a,b, we extract the mean minimum error across all simulations for the outputs that we obtain both from our learning rule (see Supplementary Fig. 5a) and from training via back-propagation (see Supplementary Fig. 5b). We then linearly interpolate between the learned gain patterns for the fast and slow outputs, and we calculate the error (see Methods Section 1.6) between the output and the target movement at the interpolated speed. We calculate these errors for many interpolated movement durations between 0.5 s and 2.5 s, and we plot the mean errors for both our learning rule and the back-propagation training in the top panel of Fig. 6d. We also show an example output that lasts 1.5 s.

**Details for Figs. 6d–f and Supplementary Fig. 6.** To demonstrate that gain modulation can provide effective smooth control of movement speed for multiple initial conditions of the neuronal activity, we train networks to generate a pair of target movements in response to a corresponding pair of orthogonal initial conditions (see the above description of Supplementary Figs. 5h–j) at fast and slow speeds and also at each of 5 intermediate, evenly-spaced speeds in between these extremes. To do this, we parametrize the gain pattern of speed index  $s$  (with  $s \in \{1, \dots, 7\}$ ) as a convex combination of a gain pattern  $\mathbf{g}_{s=1}$  for fast movements and a gain pattern  $\mathbf{g}_{s=7}$  for slow movements, with interpolation coefficients of  $\lambda_s$  (with  $\mathbf{g}_s = \lambda_s \mathbf{g}_{s=1} + (1 - \lambda_s) \mathbf{g}_{s=7}$ ,  $\lambda_1 = 1$ , and  $\lambda_7 = 0$ ). We optimize (using back-propagation, as discussed above) over  $\mathbf{g}_{s=1}$ ,  $\mathbf{g}_{s=7}$ , the 5 interpolation coefficients  $\lambda_s$  (with  $s \in \{2, \dots, 6\}$ ), and a single set of readout weights. For a given speed  $s$ , we use the gain pattern  $\mathbf{g}_s$  for both movements.

We plot the 7 learned gain patterns in Fig. 6e, and we plot their corresponding outputs for both initial conditions in Supplementary Fig. 6. (We call this collection of 7 trained gain patterns the ‘speed manifold’.) We show the linear version of the speed manifold (i.e., interpolating between the fast and slow gain patterns) in Supplementary Fig. 6b. Interpolating between the fast and slow gain patterns accurately generates both movements at any intermediate speed. (See the bottom panel of Fig. 6d.). For both initial conditions, we plot outputs at 5 evenly-spaced speeds by linearly interpolating between the fast ( $\mathbf{g}_{s=1}$ ) and slow ( $\mathbf{g}_{s=7}$ ) gain patterns in Fig. 6f.

### Simulation details for Fig. 7

We simultaneously train gain patterns for controlling different movements (i.e., different movement shapes) and their speed. We train a recurrent neuronal network (using back-propagation, as we discussed previously) to generate each of 10 different movement shapes at 7 different, evenly-spaced speeds (ranging from the fast variant to the slow variant) using a single fixed initial condition  $\mathbf{x}_0$ . To jointly learn gain patterns that control movement shape and speed, we parametrize each gain pattern as the element-wise product of a gain pattern that encodes shape (which we use at each speed for a given shape) and a gain pattern that encodes speed (which we use at each shape for a given speed). We again parametrize (see our details for Figs. 6d–f) the gain pattern that encodes the speed index  $s$  (with  $s \in \{1, \dots, 7\}$ ) as a convex combination of two common endpoints,  $\mathbf{g}_{s=1}$  (which we use for the fast-movement variants) and  $\mathbf{g}_{s=7}$  (which we use for the slow-movement



variants). We thus optimize over 10 gain patterns for movement shape, 2 gain patterns each for fast and slow movement speeds, 5 speed-interpolation coefficients (see above), and a single set of readout weights.

In Fig. 7b, we plot the gain patterns that we obtain for controlling the movement speeds at each of the 7 trained speeds. In Fig. 7c, we show the mean error between the network output and the target over the 10 target movements when generating gain patterns for movement speed by linearly interpolating between the trained fast ( $\mathbf{g}_{s=1}$ ) and slow ( $\mathbf{g}_{s=7}$ ) gain patterns. In Fig. 7d, we plot the outputs of 6 of the 10 gain patterns for movement shape at each of 5 interpolated speeds between the fast and the slow gain patterns. In rightmost panel of Fig. 7a, we plot 2 example movement shapes at 3 interpolated speeds.

### Simulation details for Fig. 8 and Supplementary Fig. 7

For these figures, we use the 10 trained gain patterns for movement shapes, as well as the speed manifold from Fig. 7 (see our simulation details for Fig. 7). Using our learning rule from Eqns. (8) and (9), we train 10 coefficients  $c_1, \dots, c_{10}$  (with one for each shape-specific gain pattern; see Fig. 8a) to construct a new gain pattern that, together with the speed manifold, generates a new target movement at the fast and slow speeds. Specifically, we replace the gains  $g_i$  (for  $i \in \{1, \dots, N\}$ ) with the coefficients  $c_i$  (for  $i \in \{1, \dots, 10\}$ ) in Eqns. (8) and (9). We use the mean of the errors at the fast and slow speeds. To generate the network output at the fast and slow speeds, respectively, we calculate the element-wise product between the newly-constructed gain pattern and the fast and slow gain pattern, respectively, on the speed manifold. We independently train, using 10,000 training iterations, the coefficients  $c_1, \dots, c_{10}$  on each of the 100 target movements that we used for Fig. 4. In Supplementary Fig. 7, we plot histograms of the errors over the 100 target movements after training for both the fast and slow speeds. We plot the mean error (see the black curve) over all 100 target movements at interpolated speeds in Fig. 8c. For the output that produces the 50th-smallest summed errors from fast and slow speeds, we plot the error in red in Fig. 8c. As a control, we calculate the mean error between the network output and the target over the 100 target movements when choosing one of the 100 newly-learned gain patterns uniformly at random without replacement. (See the grey curve in Fig. 8c.)

Additionally, instead of learning to combine gain patterns using the method that we described in the previous paragraph, we determine coefficients  $c_1, \dots, c_{10}$  using a least-squares regression by fitting the 10 learned movements to each of the 100 target movements at the fast and slow speeds simultaneously and requiring that  $c_j \geq 0$  for all  $j$  and  $\sum_{j=1}^{10} c_j = 1$ . (See the black dashed curve in Fig. 8c.)

Finally, we plot the Pearson correlation coefficient between pairs of target movements versus the Pearson correlation coefficient between corresponding pairs of learned coefficients  $c_1, \dots, c_{10}$ . In our visualization, we plot only 1,000 of the 4,950 data points. (We choose these points uniformly at random.) Note that we are unlikely to observe correlation values close to  $-1$  between pairs of combination coefficients because the coefficients  $c_1, \dots, c_{10}$  are likely to sum to approximately 1

(see our discussion of Fig. 4); in fact, we calculate the mean sum of the coefficients to be approximately 0.91.

## REFERENCES

1. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
2. G. Hennequin, T. P. Vogels, and W. Gerstner, “Optimal control of transient dynamics in balanced networks supports generation of complex movements,” *Neuron*, vol. 82, no. 6, pp. 1394–1406, 2014.
3. M. M. Churchland, J. P. Cunningham, M. T. Kaufman, J. D. Foster, P. Nuyujukian, S. I. Ryu, and K. V. Shenoy, “Neural population dynamics during reaching,” *Nature*, vol. 487, no. 7405, pp. 1–8, 2012.
4. D. Sussillo and L. F. Abbott, “Generating coherent patterns of activity from chaotic neural networks,” *Neuron*, vol. 63, no. 4, pp. 544–557, 2009.