Simple decoding of behavior from a complicated neural manifold

Sean M. Perkins [1,2], **John P. Cunningham** [2,3,4,5], **Qi Wang** [1], **and Mark M. Churchland** [2,5,6,7,✉]

[1]Department of Biomedical Engineering, Columbia University, New York, NY, USA
[2]Zuckerman Institute, Columbia University, New York, NY, USA
[3]Department of Statistics, Columbia University, New York, NY, USA
[4]Center for Theoretical Neuroscience, Columbia University Medical Center, New York, NY, USA
[5]Grossman Center for the Statistics of Mind, Columbia University, New York, NY, USA
[6]Department of Neuroscience, Columbia University Medical Center, New York, NY, USA
[7]Kavli Institute for Brain Science, Columbia University Medical Center, New York, NY, USA

Decoders for brain-computer interfaces (BCIs) assume constraints on neural activity, chosen to reflect scientific beliefs while yielding tractable computations. We document how low tangling – a typical property of motor-cortex neural trajectories – yields unusual neural geometries. We designed a decoder, MINT, to embrace appropriate statistical constraints for these geometries. MINT takes a trajectory-centric approach: a library of neural trajectories (rather than a set of neural dimensions) provides a scaffold approximating the neural manifold. Each neural trajectory has a corresponding behavioral trajectory, allowing straightforward but highly nonlinear decoding. MINT outperformed other interpretable methods, and outperformed expressive machine learning methods in 37 of 42 comparisons. Yet unlike such methods MINT's constraints are known, not the implicit result of optimizing decoder output. MINT performed well across tasks, suggesting its assumptions are generally well-matched to the statistics of neural data. Despite embracing highly nonlinear relationships between behavior and potentially complex neural trajectories, MINT's computations are simple, scalable, and provide interpretable quantities such as data likelihoods. MINT's performance and simplicity suggest it may be an excellent candidate for clinical BCI applications.

brain-computer interfaces | population geometry | neural dynamics | motor cortex | motor control

Correspondence: *mc3502@columbia.edu*

Introduction

A common goal in neural engineering and neuroscience is to estimate target variables, in real time, from observations of neural spiking activity. For brain-computer interfaces (BCIs), target variables may correspond to prosthetic motion [1–6], muscle activity [7–10], cursor control [11–17], navigation [18–20], speech [21–23], handwriting [24], or cognitive states [25–29]. In other applications, one may wish to estimate the state of the brain, a latent variable, for clinical monitoring [30], data visualization [31], or closed-loop neurally contingent experiments [32]. Spiking activity is high-dimensional (dimensionality matching the number of neurons) and can be variable across repeated trials of a nominally identical behavior. This complexity can be managed by utilizing statistical models that simplify the assumed relationship between spiking activity and variables of interest. For example, early BCI decoders estimated horizontal and vertical cursor velocity via weighted sums of neural activity, allowing velocity to be readily decoded even for spike patterns that were never observed during training. Ideally, estimation should leverage assumptions that are both accurate and strongly simplifying.

Perhaps the most common constraint assumed by models of spiking activity is the notion of 'rate'. Rates are estimated in various ways by binning, filtering, and/or trial-averaging. Fundamentally, a neuron's rate is a value determining its instantaneous probability of spiking. Statistically, this assumption has been powerful, especially in brain areas where spiking shows considerable variability. Furthermore, in artificial spiking networks, rate is a well-defined quantity whose properties justify those statistical models [33]. It is thus common practice to assume that neural activity, at each moment in time, can be summarized by a vector of continuously varying rates with one entry per neuron (the 'neural state'). Spikes are then treated as noisy observations of the neural state.

A variety of methods for estimating the neural state and decoding behavior on single trials have been proposed

[34–40]. Many of these leverage an additional assumption: that the neural state is low-dimensional, with the vector of rates exploring far fewer dimensions than the number of recorded neurons. Low-dimensional state estimation and decoding are often implicitly combined – e.g. linear decoders consider a neural state whose dimensionality matches the number of decoded variables [1–3, 5, 11, 12].

Many studies have usefully assumed further constraints on the neural state, beyond those related to dimensionality. Multiple studies have assumed constraints on neural activity related to structure in behavior [13, 16, 17, 41–49]. Studies have leveraged assumptions regarding linear [37] or nonlinear [38] neural dynamics that constrain how the neural state can evolve. Sani et al. [50] leveraged the assumption that only a subspace within the low-dimensional neural state is relevant to behavior. Nonlinear decoders (including neural networks) can relax some constraints (e.g. linear relationships between behavior and neural state) while imposing or learning others [21, 23, 24, 38, 40, 51–60].

Each of these prior approaches succeeded by imposing structure on the estimation problem via constraints (explicit or learned) on the neural state or its relationship to behavior. Here, we propose a novel algorithm, Mesh of Idealized Neural Trajectories (MINT), that leverages different constraints. These constraints derive from advances – some recent and some presented here – in our understanding of the geometry of population activity and its relationship to movement. MINT takes a trajectory-centric view of neural geometry, in which the manifold of observable neural states is best approximated using stereotyped trajectories and interpolations between them. MINT assumes that the relationship between neural activity and behavior, while highly nonlinear, can be approximated by creating a correspondence between neural and behavioral trajectories. MINT is well-suited to situations where neural activity is shaped by dynamics, yet makes no assumption regarding the form of those dynamics. Because MINT's assumptions were chosen to respect scientific beliefs rather than tractability, they might be expected to complicate computation. Fortuitously the opposite is true: MINT's assumptions make it easier to compute desired quantities such as data likelihoods.

Below we document and quantify properties of motor-cortex population geometry – and its relationship with behavior – that are relevant to any decoder (and more generally to many scientific analyses and hypotheses). We describe how MINT was designed to respect and leverage these properties. We compare MINT's decode performance with multiple other decode methods. We do so across a variety of tasks and behavioral variables, to document MINT's flexibility. We also evaluate MINT as a method for neural-state estimation. MINT's overall high level of performance illustrates the value of a trajectory-centric view that respects the empirical structure of population geometry.

Results

Neural trajectories are sparsely distributed, stereotyped, and can be high-dimensional

We analyzed 8 datasets consisting of simultaneous neural recordings and behavioral measurements from primates performing motor, sensory, or cognitive tasks. To begin, we focus on one dataset (MC_Cycle) recorded during a task where a primate grasps a hand-pedal and moves it cyclically forward or backward to navigate a virtual environment. We use this dataset to document some unusual properties of motor-cortex population geometry – properties that have implications for any BCI-decoding method. A variety of studies indicate that the dominant features of neural trajectories do not reflect encoded parameters or network outputs, but rather the computational needs of network solutions (e.g. [61, 62]). For example, noise-robust solutions require low trajectory tangling [63], resulting in neural trajectories that look very different from the outputs they create [64].

By definition, low-tangled neural trajectories never come close to crossing while traveling in different directions. For example, forward-cycling trajectories and backward-cycling trajectories appear to overlap when viewed in two-dimensions (Fig. 1a, *top*). Yet the apparent crossings do not result in high trajectory tangling; there exist other dimensions where forward and backward trajectories are well separated (Fig. 1a, *bottom*, same scale used for both). Separation keeps trajectory tangling extremely low – far lower than for the corresponding muscle trajectories.

Under the commonly assumed low-dimensional constraint, the manifold of observable neural states is equivalent to a subspace. This assumption is so standard that the terms 'manifold' and 'subspace' have often been used interchangeably. From that perspective, all states within that subspace are possible, with probabilities that may be described by a covariance matrix. For example, if two states are likely, the state between them is also likely. A family of low-tangled trajectories will often display the opposite property. Low-tangled trajectories typically require large voids that are never occupied. For example, circular trajectories are common [62, 63, 65]. Their intrinsically low tangling would increase if other trajectories (e.g. from other conditions) intruded into their center, but empirically this is avoided. Thus, the manifold of observable states may be far more sparsely distributed than is suggested by the simple constraint of a subspace or covariance matrix.

Maintaining low trajectory tangling can require adding 'new' dimensions to separate trajectories. Thus, neural-state dimensionality may be higher than the number of variables one wishes to decode [66, 67]. Furthermore,

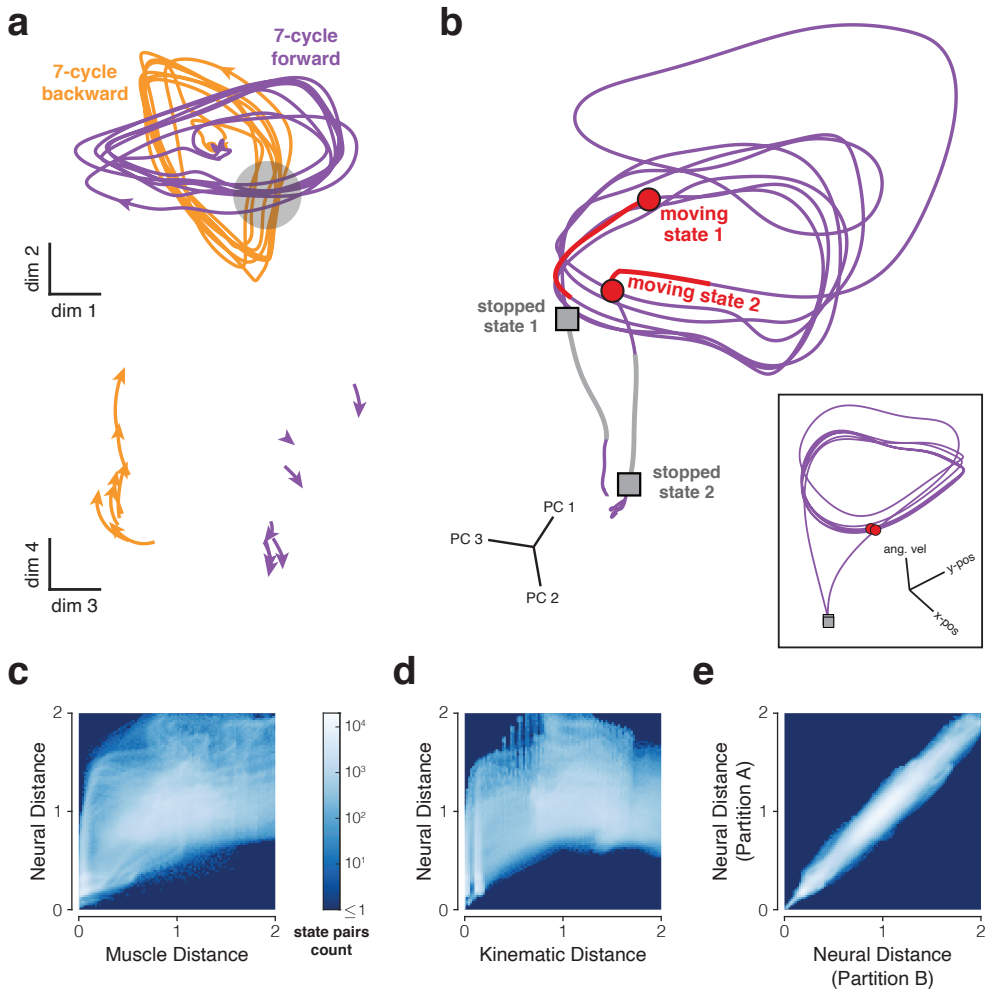

Figure 1. Properties of neural trajectories in motor cortex, illustrated for data recorded during the cycling task (MC_Cycle dataset). a) Low tangling implies separation between trajectories that might otherwise be close. *Top.* Neural trajectories for forward (*purple*) and backward (*orange*) cycling. Trajectories begin 600 ms before movement onset and end 600 ms after movement offset. Trajectories are based on trial-averaged firing rates, projected onto two dimensions. Dimensions were hand-selected to highlight apparent trajectory crossings while also capturing considerable variance (11.5%, comparable to the 11.6% captured by PCs 3 and 4). *Gray region* highlights one set of apparent crossings. *Bottom.* Trajectories during the restricted range of times in the gray region, but projected onto different dimensions: the top two principal components when PCA was applied to this subset of data. The same scale is used in top and bottom subpanels. **b)** Examples of similar behavioral states (*inset*) corresponding to well-separated neural states (*main panel*). Colored trajectory tails indicate the previous 150 ms of states. Data from 7-cycle forward condition. **c)** Joint distribution of pairwise distances for muscle and neural trajectories. Analysis considered all states across all conditions. For both muscle and neural trajectories, we computed all possible pairwise distances across all states. Each muscle state has a corresponding neural state, from the same time within the same condition. Thus, each pairwise muscle-state distance has a corresponding neural-state distance. The color of each pixel indicates how common it was to observe a particular combination of muscle-state distance and neural-state distance. Muscle trajectories are based on seven z-scored intramuscular EMG recordings. Correspondence between neural and muscle state pairs included a 50 ms lag to account for physiological latency. Results were not sensitive to the presence or size of this lag. Neural and muscle distances were normalized (separately) by average pairwise distance. **d)** Same analysis for neural and kinematic distances (based on phase and angular velocity). Correspondence between neural and kinematic state pairs included a 150 ms lag. Results were not sensitive to the presence or size of this lag. **e)** Control analysis to assess the impact of sampling error. If two sets of trajectories (e.g. neural and kinematic) are isometric and can be estimated perfectly, their joint distribution should fall along the diagonal. To estimate the impact of sampling error, we repeated the above analysis comparing neural distances across two data partitions, each containing 15-18 trials/condition.

low trajectory tangling requires that two similar neural states lead to similar future neural states, even over timescales longer than those governing temporal smoothness. Thus, sequences of neural states over time may be stereotyped. Recent work used this assumption to compare network and biological solutions [68]. Stereotyped trajectories would also be consistent with recent BCI-adaptation results. BCI adaptation involves reusing existing patterns of neural activity [69], and trajectories do not reverse even when that would be advantageous [70, 71].

These observations argue that the appropriate constraints on the neural state have little to do with dimensionality per se – dimensionality may actually be quite high. Rather, constraints should reflect that neural states are sparsely distributed and belong to families of stereotyped trajectories. MINT makes assumptions that embrace these aspects of state-trajectory geometry. We briefly summarize these assumptions here, leaving a detailed description of MINT for a later section. First, MINT accommodates any dimensionality (high or low) by operating directly on the neural state rather than attempting to reduce its dimensionality. Second, MINT assumes the manifold of observable states is extremely sparsely distributed and therefore best approximated by leveraging the trajectories themselves rather than their covariance matrix or corresponding subspace. Third, MINT assumes that neural states are traversed in a stereotyped order (i.e. each neural state determines the subsequent state). MINT embodies these assumptions by approximating the manifold of observable neural states – and the transitions between them – with a 'mesh' of neural trajectories. A library of idealized trajectories samples the underlying manifold. Interpolations between neural states connect these trajectories into a mesh that spans the manifold more completely, allowing MINT to decode neural and behavioral states beyond those observed during training.

Neural and behavioral trajectories are non-isometric

The geometry of neural trajectories also has implications regarding how the neural state, once it is estimated, should be decoded into a behavioral state. To explore, we compared distances in neural and behavioral space. Well-separated neural states frequently corresponded to the same behavioral state. For example, in Figure 1b, the two red circles correspond to moments when behavior is essentially identical: cycling at $\sim$1.7 Hz with the hand approaching the 'bottom' of the cycle. Nevertheless, the corresponding moments are distant in neural state space. Similarly, the two gray squares correspond to two moments when the hand is stopped at the bottom (immediately before and after bouts of cycling) and are also quite distant. The neural and kinematic dimensions in Figure 1b were selected and oriented to highlight some resemblance between neural and behavioral trajectories. Yet, there are geometric differences even in this view, and these would only become more pronounced if more conditions (e.g. backward cycling) were included. As a consequence, the neural-to-behavioral-state mapping is many-to-one.

To compare trajectory-geometries quantitatively, we leveraged the fact that each neural distance has a corresponding behavioral distance, taken for the same pair of conditions and times (allowing for a latency shift). If geometries are similar, behavioral and neural distances should be strongly correlated. To establish a baseline that takes into account the impact of sampling error, we compared neural distances across two partitions. As expected, nearly all data fell along the diagonal (Fig. 1e). The joint distribution was strikingly non-diagonal when comparing neural versus muscle-population trajectories (Fig. 1c) and neural versus kinematic trajectories (Fig. 1d).

Thus, neural trajectories are isometric to neither muscle nor kinematic trajectories. Similar behavioral states frequently corresponded to dissimilar neural states. Defining similar kinematic states to be those with normalized distance <0.1, corresponding neural states were as close as 0.02 and as far as 1.46. Nevertheless, dissimilar behavioral states did not correspond to similar neural states. Defining dissimilar kinematic states to be those within 0.1 of distance 2, neural states were never closer than 0.53. These relationships facilitate decoding in some ways – different behavioral states imply different neural states – while also creating a challenge – decoding must map dissimilar neural states to similar behavioral states.

In principle, this many-to-one mapping could be accomplished linearly. Suppose neural trajectories mirrored behavioral trajectories, but with additional non-behavior-encoding dimensions. A linear decoder could simply ignore those additional dimensions. Yet this will be suboptimal if the ignored dimensions capture the majority of response variance. MINT leverages all dimensions by learning, directly from training data, which behavioral state is associated with each neural state in a library of idealized trajectories. This allows highly nonlinear mappings – including many-to-one mappings – without needing to choose a nonlinear function class and fit it to the data.

Training and decoding procedure

For training, MINT requires a library of neural trajectories (Ω) and a corresponding library of behavioral trajectories (Φ). Libraries can be learned in a variety of ways, depending on the nature of the task and training data. Because trial-averaging is particularly easy to apply, we focus first on libraries of neural trajectories learned in this way.

Consider a center-out reaching task. Neural and behavioral trajectories (Fig. 2a) have different geometries: stacked loops versus radially arranged and relatively straight. Trajectories are learned from a training set containing repeated reaching trials to each target location. Each moment in the training data corresponds to a condition (c)

and an index (k) indicating progress through the movement. For example, c = 3 corresponds to the purple reach condition and k = 240 indicates a moment 240 ms into that movement. There will typically be many such moments in the training data: one for each trial of that type. Averaging across trials yields a behavioral state $\overline{\mathbf{z}}_k^c$. Similarly, temporally filtering spikes and averaging across trials yields a neural state $\overline{\mathbf{x}}_k^c$, containing each neuron's estimated rate. Rate estimation can be further improved by a variety of means, including additional smoothing across neurons and/or conditions (see Methods). For some tasks, training data may not be organized by discrete conditions. Rates would then be estimated using methods devised for that purpose (e.g. [38, 40]).

The library of neural trajectories (Ω) and the corresponding library of behavioral trajectories (Φ) are the parameters of MINT. They specify which states can be occupied and the order in which they are traversed. Prior work has utilized stereotyped behavioral trajectories [41, 46] or a mixture of condition-specific behavioral trajectory models that capture trial-by-trial movement variability [49]. For tractability, those methods assumed low-dimensional neural states that are roughly isometric to arm velocities. With MINT, neural trajectories can take any geometry, are learned empirically, and are related to behavioral trajectories via c and k, rather than by an assumed geometric isometry. The relationship between neural and behavioral states can be highly nonlinear and many-to-one. Although in Figure 2a there is one trajectory per target location, in practice there may be multiple trajectories per target (e.g. different speeds or curvatures) to enrich the library by allowing for movement variations. At the same time, it is not necessary for every potential target location and movement variation to be present in the training set. As will be discussed below, MINT can generalize in a handful of ways, including between any two conditions where neural activity and behavior both change smoothly. Furthermore, although the design of MINT is inspired by observations related to low tangling during movement, it is perfectly acceptable if there are moments of high tangling within Ω. Indeed this often occurs at transitions, e.g. from not-yet preparing to preparing a specific reach.

During decoding, incoming observations of spiking activity are binned (Fig. 2b, Δ = 20 ms for all analyses here). Suppose we are at time-bin t' within a session where behavior is being decoded. Recent spiking observations are denoted by $\overline{\mathbf{s}}_{t'-\tau':t'}$, where τ' specifies the number of previous time bins that are retained in memory. We compute the log-likelihood that we would have observed $\overline{\mathbf{s}}_{t'-\tau':t'}$ given each possible neural state, under a Poisson spiking model (Fig. 2c). The set of possible neural states is initially restricted to the states within the library of neural trajectories. This is a strong, but important assumption. If one were to consider all possible neural states in a 100-dimensional space, with each rate discretized into 10 possible values, it would be necessary to compute log-likelihoods for 10^{100} potential neural states. Yet computing log-likelihoods becomes tractable if trajectories are sparsely distributed. For example, even with 1000 samples per neural trajectory for the 4 neural trajectories in Figure 2, there would still be only 4000 possible neural states for which log-likelihoods must be computed (in practice it is fewer still, see Methods). Beyond reducing computational complexity, MINT's assumptions regarding where the neural state can (and can't) reside should aid decoding if those assumptions are well-aligned with the data.

MINT also leverages the order in which states are traversed, providing another massive reduction in complexity. MINT assumes that each neural state along each neural trajectory has a unique recent history that can be compared to the recent history of spike counts when computing log-likelihoods. Fortuitously, spike counts in non-overlapping bins are conditionally independent given rate in a Poisson spiking model. Thus, the log-likelihood associated with a particular neural state decomposes across time bins and neurons into Poisson log-likelihoods (which can be queried from a precomputed lookup table). Furthermore, many terms computed for decoding in one time bin can be reused at the next time bin. A recursive solution for exploiting this redundancy reduces the complexity of computing these log-likelihoods, leading to a lightweight procedure amenable to real-time applications. The assumption of stereotypy trajectory also enables neural states (and decoded behaviors) to be updated in between time bins. While waiting for the next set of spiking observations to arrive, MINT simply assumes that each neural state advances deterministically along the trajectory to which it is assigned.

Using the above procedure, one could simply take the maximum-likelihood neural state from the library, then render a behavioral decode based on the corresponding behavioral state. However, we wish the trajectory library to serve not as an exhaustive set of possible states, but as the basic structure defining a 'mesh' of possible states. Suppose the subject made a reach that split the difference between purple and blue reach conditions in Figure 2a. MINT would identify two candidate states with high likelihoods, along different neural trajectories, and interpolate (Fig. 2d). A line segment connecting those states is parameterized by α, ranging from 0 (purple state) to 1 (blue state). Because the log-likelihood of the observed spikes is a concave function of α, the optimal neural state can be rapidly identified online using Newton's method. The same interpolation produces the final decode from the corresponding behavioral states (Fig. 2e).

The 'mesh' in MINT conveys that interpolation fills in the space between candidate states along different trajectories. The mesh is not fully defined during training because it is unknown which candidate states to consider until the spiking observations arrive. Rather, as spikes are observed during decoding, they will often be consistent with more than one trajectory. It is at that juncture that MINT utilizes interpolation to consider a space of intermediate trajectories that could have given rise to these observations. It is easiest to think of the mesh as connecting nearby

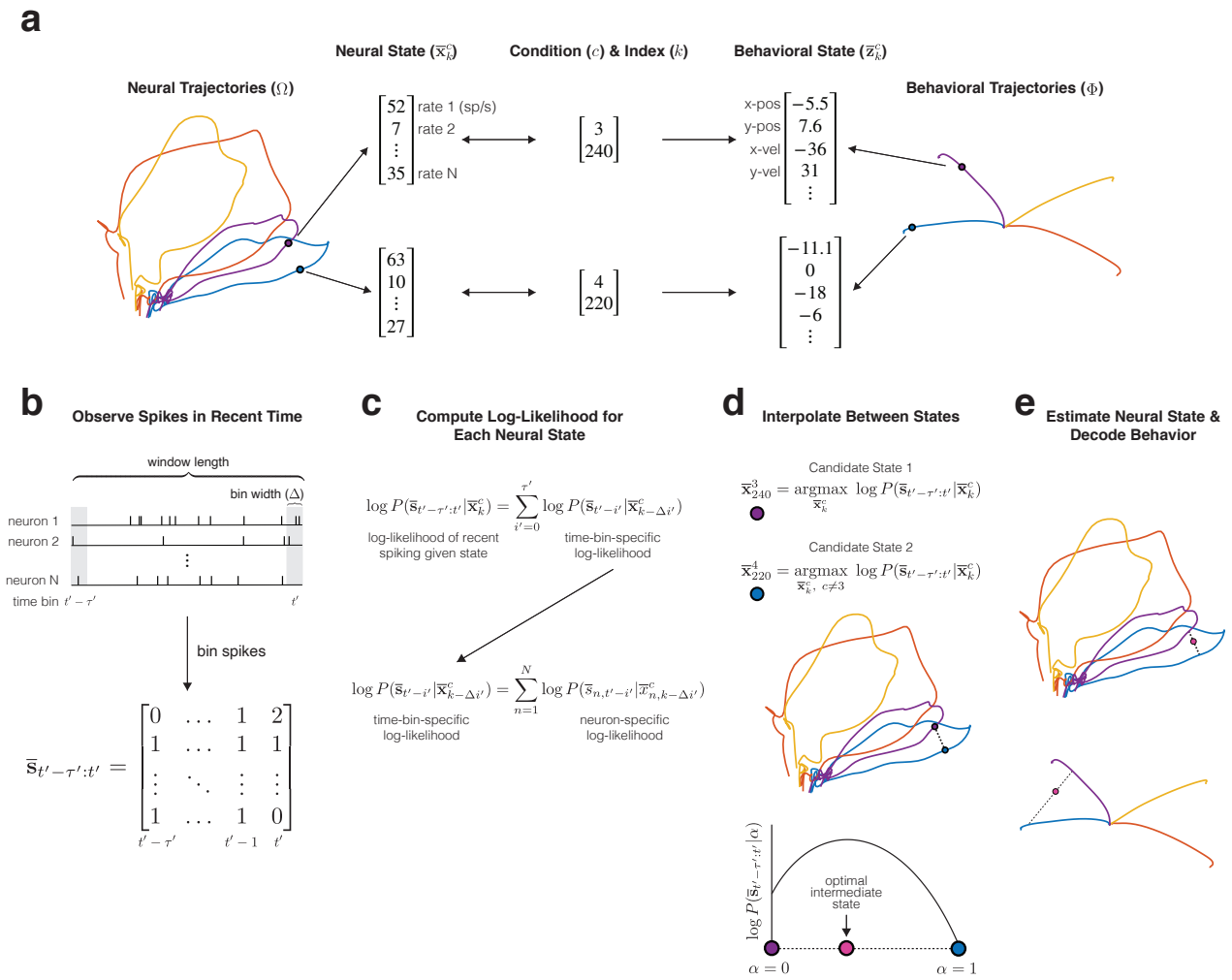

Figure 2. Example training (top panel) and decoding (bottom panels) procedures for MINT, illustrated using four conditions from a reaching task. a) Libraries of neural and behavioral trajectories are learned such that each neural state $\overline{\mathbf{x}}_k^c$ corresponds to a behavioral state $\overline{\mathbf{z}}_k^c$. **b)** Spiking observations are binned spike counts (20 ms bins for all main analyses). $\overline{\mathbf{s}}_{t'-\tau':t'}$ contains the spike count of each neuron for the present bin, t', and τ' bins in the past. **c)** At each time step during decoding, the log-likelihood of observing $\overline{\mathbf{s}}_{t'-\tau':t'}$ is computed for each and every state in the library of neural trajectories. Log-likelihoods decompose across neurons and time bins into Poisson log-likelihoods that can be queried from a precomputed lookup table. A recursive procedure (not depicted) further improves computational efficiency. Despite utilizing binned spiking observations, log-likelihoods can be updated at millisecond resolution (Methods). **d)** Two candidate neural states (*purple* and *blue*) are identified. The first is the state within the library of trajectories that maximizes the log-likelihood of the observed spikes. The second state similarly maximizes that log-likelihood, with the restriction that the second state must not come from the same trajectory as the first (i.e. must be from a different condition). Interpolation identifies an intermediate state that maximizes log-likelihood. **e)** The optimal interpolation is applied to candidate neural states – yielding the final neural-state estimate – and their corresponding behavioral states – yielding decoded behavior.

trajectories, as in Figure 2d,e, and this is indeed what occurs in practice. While no assumption prevents MINT from interpolating between very different trajectories, this is empirically rare: the vast majority of decoded states are near an on-trajectory state. For on-trajectory states, direct decoding should be nearly perfect in determining the associated behavior. For states near trajectories, linear interpolation should provide good generalization because neural activity and behavior typically vary together smoothly.

Behavioral decoding in a multi-task network

MINT can be used in the same way for different tasks – the libraries differ but all other operations remain the same. If training data include multiple tasks, MINT can readily and implicitly switch tasks 'on the fly' based only on neural observations. MINT can do so even when the nonlinear relationship between neural activity and behavior is very different across tasks. If desired, MINT can report which type of behavior is presently being produced and decode

only those variables relevant to that behavior. This could be useful, for example, if a subject were controlling both a computer cursor and a wheelchair. It is presently rare for physiological data to be recorded across multiple tasks in the same session. We thus demonstrate this task-switching ability for an artificial spiking network that performs two tasks on interleaved trials. This evaluation is also useful because the ground-truth neural state is known.

We evaluated MINT's performance on data produced by an artificial recurrent network built from leaky-integrate-and-fire spiking neurons [33]. Network neurons exhibit realistic firing rates and realistic levels of spiking variability (i.e. spiking statistics are roughly Poisson). The network was trained to generate posterior deltoid activity (Fig. 3a) for a reaching task (similar to that in Fig. 2) and a cycling task (similar to that in Fig. 1). The network uses different strategies to perform the two tasks, and neural activity occupies distinct subspaces for each (Fig. 3b). This provides a stringent challenge; correlations between neural activity and behavioral output are completely different across tasks.

Decoding performance was excellent: R^2 = .968 over $\sim$7.1 minutes of test trials. Decoded deltoid activity (*green*) was virtually identical to network output (*black*). This was true during both reaching (R3, R4 and R6 correspond to three reach directions) and cycling (C), with smooth transitions between them. MINT's decoding approach was not hindered by the very different relationship between neural trajectories and behavioral output during the two tasks, nor by the dramatically different neuron-neuron covariance. Rather, MINT simply identified the most likely neural state at each moment and decoded the associated behavioral output.

The fact that the ground truth is known in this network illuminates a challenge that may impact decoding in many real-world situations. Even when a neural population or brain area exists to generate a particular output, that output may be encoded in low-variance dimensions and it may therefore be difficult to decode unless other dimensions can also be leveraged. For the present network, the output is simply a weighted sum of spikes, across all 1200 neurons. Thus, a linear readout (learned via ridge regression) that leveraged all 1200 neurons decoded network output exceptionally well (R^2 = .982). However, the output dimension in this network captures exceedingly little of the total variance of network responses ($\sim$2%), and thus is unlikely to perform well if only a subset of neurons were recorded. Indeed, when only using 5% of neurons in the network, linear decoding performance dropped to R^2 = .773. In contrast, MINT's performance only dropped by .001 to R^2 = .967. Thus, even when the ground-truth readout is known, MINT can provide improved performance by leveraging all neural dimensions, not only those that directly impact behavioral output. Furthermore, MINT can readily decode other variables (as will be shown below) that are unlikely to be directly controlled by linear sums of spiking activity.

Behavioral decoding across multiple datasets

We investigated how MINT performed on four datasets in which primates performed a variety of motor tasks. All tasks involved simultaneously collected spiking activity and behavioral measurements. MC_Cycle is the same dataset from Figure 1. MC_Maze involves neural recordings from primary motor cortex (M1) and dorsal premotor cortex (PMd) during straight and curved reaches [72]. Area2_Bump involves neural recordings from area 2 of somatosensory cortex during active and passive center-out-reaches [73]. MC_RTT involves M1 recordings during point-to-point reaches without condition structure [58]. All decoding was offline, to allow evaluation across a variety of task types and comparison with other decode methods (below) across multiple decoded variables. All decoding was causal; i.e. based on spikes from before the decoded time. All decoding examples and performance quantifications utilized held-out test sets. Discussion of how MINT's (few) hyperparameters were selected for the analyses in Figure 3 will be provided in the next section.

Decoding closely tracked actual behavior (Fig. 3c). For the MC_Cycle dataset, MINT closely tracked pedal phase and angular velocity. Decoding was virtually lag-free (on average): phase was decoded with near-zero lag, and transitions between rest and cycling occurred with near-zero latency. This was true despite the fact that all decoding was causal. During periods of non-movement, decoded angular velocity remained flat with few false movements. To illustrate, consider a 70 second stretch of cycling (Supp. Video) that included $\sim$30 seconds of true non-movement (true angular speed <.05 Hz). Small decode mistakes (decoded angular speed >0.1 Hz) did occasionally occur (e.g. Fig. 3c, small 'blip' before the middle cycling bout). Yet such mistakes were rare: decoded angular speed was <0.1 Hz for $\sim$99% of non-moving times.

A subtle point is that MINT assumes stereotyped trajectories when computing data likelihoods, but this does not cause the maximum-likelihood state to follow a stereotyped trajectory during decoding. For example, individual trials rarely have a cycling speed that perfectly matches the library trajectory. Yet this does not cause a growing discrepancy between decoded and actual phase. If cycling is slightly faster than the library trajectory, MINT advances along that trajectory at a faster-than-usual rate. This flexibility is essential to good performance, but also reveals itself during errors when behavioral trajectories exhibit features (e.g. the small blip noted above) not present in any library trajectory.

In the MC_Maze dataset, decoded velocity tracked actual reach velocity across a wide variety of straight and curved reaches (108 conditions). Decoded and actual velocities were highly correlated (R_x = .962, R_y = .950). We

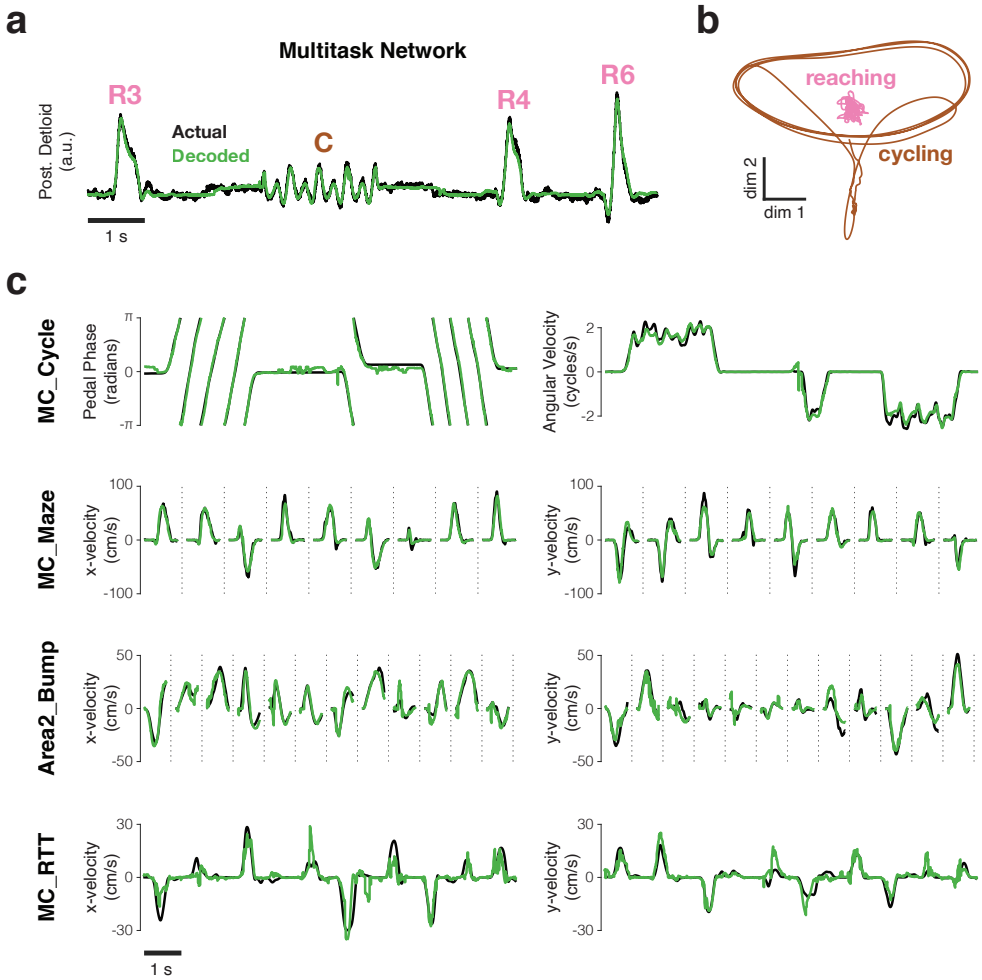

Figure 3. Examples of behavioral decoding provided by MINT for one simulated dataset and four empirically recorded datasets. All decoding is causal; only spikes from before the decoded moment are used. **a)** MINT was applied to spiking data from an artificial spiking network. That network was trained to generate posterior deltoid activity and to switch between reaching and cycling tasks. Based on spiking observations, MINT approximately decoded the true network output at each moment. 'R3', 'R4', and 'R6' indicate three different reach conditions. 'C' indicates a cycling bout. MINT used no explicit task-switching, but simply tracked neural trajectories across tasks as if they were conditions within a task. **b)** Illustration of the challenging nature, from a decoding perspective, of network trajectories. Trajectories are shown for two dimensions that are strongly occupied during cycling. Trajectories for the 8 reaching conditions (*pink*) are all nearly orthogonal to the trajectory for cycling (*brown*) and thus appear compressed in this projection. **c)** Decoded behavioral variables (*green*) compared to actual behavioral variables (*black*) across four empirical datasets. MC_Cycle and MC_RTT show 10 seconds of continuous decoding. MC_Maze and Area2_Bump show randomly selected trials, demarcated by *vertical dashed lines*.

quantified error in absolute terms using the mean absolute error (MAE). Absolute errors were low (MAE_x = 4.6 cm/s, MAE_y = 4.8 cm/s) relative to the range of x- and y-velocities (224.8 and 205.7 cm/s, respectively).

Decoding performance was also high for Area2_Bump (R_x = .939, R_y = .895; MAE_x = 4.1 cm/s, MAE_y = 4.4 cm/s). Errors were slightly smaller than for the MC_Maze dataset in absolute terms, but were modestly larger in relative terms because the behavioral range of velocities was smaller. Errors are thus slightly clearer in the traces plotted in Fig. 3c. The overall high performance – and minimal lag – is notable given that the decode was based on spiking in a primary sensory area.

The task used in the MC_RTT dataset was not designed to include repeated trials of the same movement. Thus, when learning neural trajectories, we used a method that doesn't rely on averaging: AutoLFADS [39]. This yielded one long neural trajectory during training (only broken by recording gaps). MINT's interpolation procedure was modified to 1) occur across states >1000 ms apart, and 2) perform multiple pairwise interpolations between candidate-state pairs and use that which yielded the highest log-likelihood. The first modification ensures interpolation must consider different movements, even without a condition structure. The second modification accommodates the fact that this less-constrained task has many degrees of behavioral freedom. There may thus be more than two high-probability candidate states that are worth considering.

This strategy yielded decoded velocities (Fig. 3c) that correlated well (R_x = .786, R_y = .843) with actual velocities, but less well than for the other three datasets. Absolute errors were lower than for the other reaching datasets (MAE_x = 2.7 cm/s, MAE_y = 2.1 cm/s). However, x- and y-velocity ranges were also lower (86.6 and 61.3 cm/s). As will be documented below, decode performance for the MC_RTT dataset was lower, relative to that for the other three datasets, not only for MINT but also for other decode algorithms. There are two likely reasons. First, MC_RTT contained by far the slowest reaches of the three datasets. When analyzing only movement periods (reach speed >1 cm/s), the median speed across all movement times was 38.1, 16.8, and 5.9 cm/s for MC_Maze, Area2_Bump, and MC_RTT, respectively. Second, the ability of AutoLFADS to denoise trajectories based on inferred dynamics may have been limited by the relatively short (8.1 minutes) amount of training data. This issue aside, the MC_RTT task provides a useful demonstration that the neural trajectories needed to parameterize MINT can be obtained without a standard condition structure and without trial averaging.

Comparison to other decoders

We compared MINT to four other decode algorithms: the Kalman filter, Wiener filter, feedforward neural network, and recurrent neural network (GRU). The Kalman filter and Wiener filter are historically popular BCI algorithms [1, 43] that are computationally simple and make linear assumptions. The feedforward neural network and GRU are expressive nonlinear function approximators that have been proposed for neural decoding [52, 74] and can be implemented causally. At each decoding time, both the feedforward neural network and the GRU leverage recent spiking history. The primary difference between the two is whether the networks process that history all at once (feedforward network) or sequentially (GRU).

Each method was evaluated on each of four datasets, generating predictions for behavioral variables on held-out test sets (Fig. 4). All decoders were provided with a trailing window of spiking observations (binned in 20 millisecond increments). The length of this trailing window was optimized separately for each decoder and dataset (and, like all hyperparameters, was chosen using validation data). For non-MINT decoders, hyperparameters were tuned, using Bayesian optimization, to maximize performance [75]. MINT's few hyperparameters were less extensively optimized. For example, window length was optimized once per dataset, rather than separately for different sets of behavioral variables (as was done for the non-MINT decoders). This choice embraces an unusual and useful aspect of MINT: there is no need to retrain across behavioral variables. Once the neural state has been estimated, all behavioral variables are readily and equally decodable. Less-extensive optimization is unlikely to have put MINT at a meaningful disadvantage; MINT is robust across reasonable hyperparameters choices (Fig. S1).

By necessity and desire, all comparisons were made offline, enabling benchmarked performance across a variety of tasks and decoded variables, where each decoder had access to the exact same data and recording conditions. There were 15 total 'behavioral groups' across the four datasets. For example, MC_Maze yielded two behavioral groups, position and velocity, each of which had horizontal and vertical components.

MINT yielded the best performance, of all decoders, for 11/15 behavioral groups. For 3 of the 4 behavioral groups where MINT's performance was not the highest, the deficit relative to the best method was very small: $\Delta R^2 < .004$. We first consider position and velocity, two variables commonly focused on in BCI decoding. For position, three decoders – MINT, the GRU, and the feedforward network – provided extremely similar and very accurate performance. For velocity, MINT provided a small but noticeable improvement over the GRU and the feedforward network for two datasets (MC_Maze and Area2_Bump), was modestly worse for one dataset (MC_RTT), and was involved in essentially a three-way tie for one dataset (MC_Cycle). MINT tended to display the largest improvements, relative to the next-best decoder, when decoding muscle-related variables (EMG, force, muscle velocity, and muscle length) and phase (circular variables can be challenging for some methods but are not for MINT).

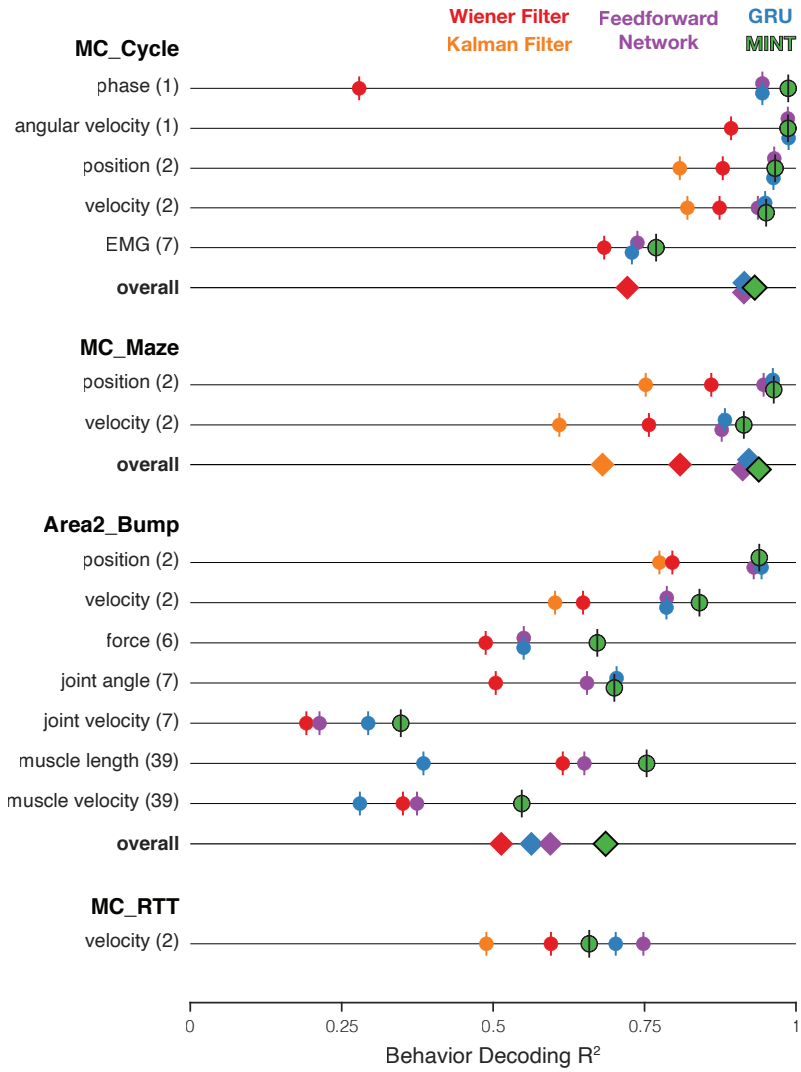

Figure 4. Comparison of decoding performance, for MINT and four additional algorithms, for four datasets and multiple decoded variables. On a given trial, MINT decodes all behavioral variables in a unified way based on the same inferred neural state. For non-MINT algorithms, separate decoders were trained for each behavioral group. E.g. separate GRUs were trained to output 'position' and 'velocity' in MC_Maze. Parentheticals indicate the number of behavioral variables within a group. E.g. 'position (2)' has two components: x- and y-position. R^2 is averaged across behavioral variables within a group. 'Overall' plots performance averaged across all behavioral groups. R^2 for feedforward networks and GRUs are additionally averaged across runs for 10 random seeds. The Kalman filter is traditionally utilized for position- and velocity-based decoding and was therefore only used to predict these behavioral groups. Accordingly, the 'overall' category excludes the Kalman filter for datasets in which the Kalman filter did not contribute predictions for every behavioral group. Vertical offsets and vertical ticks are used to increase visibility of data.

Relative to the other computationally 'lightweight' methods – the Wiener and Kalman filter – MINT's performance was consistently better, often by a large margin.

For each dataset, we computed overall performance by simply averaging across behavioral groups. In three of the four datasets (all but MC_RTT, discussed further below), overall performance was highest for MINT. Thus, despite using simple and interpretable operations, MINT is extremely competitive with highly expressive machine learning methods. Within these three datasets (MC_Cycle, MC_Maze, Area2_Bump), MINT performed well for every individual behavioral group. This consistency reflects a feature noted above: once the neural state is estimated, it becomes possible to decode any behavioral variable that has a relationship with the neural state (though some, such as EMG, may of course be intrinsically 'noisier').

MINT's lowest performance, relative to other decoders, occurred for the MC_RTT dataset. MINT outperformed the Kalman filter and Wiener filter. Yet improved decoding was achieved by the feedforward network and GRU. Thus, this was the only situation, across all datasets and behavioral groups, where MINT was ever at a meaningful performance disadvantage. A likely explanation is that the training data in the MC_RTT task are not ideally suited for estimating the library of trajectories, limiting subsequent performance on test data. Thus, if using MINT is a goal for tasks of this type, more structured training data (e.g. repeated trials of the same type) might be helpful. Alternatively, it may be sufficient simply to collect more training data. This dataset contains only 8.1 minutes of training data, which may limit the ability of AutoLFADS to estimate single-trial rates. If collecting more training data is impractical, there exist methods that improve estimation of single-trial rates (and thus trajectories) based on data from other sessions using the same task [38]. We do not pursue these avenues here, but simply stress that the principal limitation on MINT's performance is likely to be the ability to estimate an accurate library of trajectories. When doing so is challenging, MINT may be at a disadvantage. Conversely, if a library can be accurately estimated, MINT is likely to perform well across a broad range of decoded variables. In this way, MINT can be seen as largely reducing the problem of decoding to a problem of estimating a library of trajectories.

When is generalization possible across tasks?

A natural hope of BCI decoding is to find a mapping between neural and behavioral states that generalizes across tasks, eliminating the need for task-specific training data. Whether this is achievable is not known; a basic challenge is that neural activity can sometimes occupy different subspaces across tasks. To explore the implications of such geometry, we leveraged the MC_Cycle dataset. Neural activity during forward and backward cycling typically occurs in rather different subspaces [63]. In the dataset used here, those subspaces are quite close to orthogonal (principal angles of 88° and 73° between the top two principal components for forward steady-state cycling and the top two principal components for backward steady-state cycling). When asked to generalize across cycling directions, all decode methods (including MINT) generalized poorly. Across all behavioral groups and all decode methods, generalization R^2 values were typically negative and the largest positive value was 0.042. This was true even though the marginal distributions of x- and y-position and x- and y-velocity were similar across cycling directions.

The failure of all methods makes sense given the near-orthogonality of forward and backward neural trajectories (Fig. S2). MINT has no decoding advantage in this regard, but its focus on the geometry of neural activity may make it easier to anticipate when generalization is and isn't likely to be possible. When neural trajectories are nearly orthogonal to those learned during training, the existing 'mesh' will not support generalization. This knowledge allows such situations to be recognized and handled appropriately.

Modeling & preprocessing choices

MINT uses a direct mapping from neural states to behavioral states and uses interpolation to potentially improve decoding. To determine the impact of these choices, we ran a full factorial analysis comparing the direct mapping to a linear mapping (Fig. S3a), and assessing the impact of interpolation (Fig. S3b). Both choices did indeed improve performance. The analysis also assessed how acausal decoding performed relative to causal decoding (Fig. S3c). Causal decoding is a requirement for real-time BCIs, and thus is employed for all our main decoding analyses. However, future applications could potentially introduce a small lag into online decoding, effectively allowing some 'acausal decoding' at the expense of longer-latency responses. As expected, acausal decoding provided modest performance increases, raising the possibility of an interesting tradeoff. Decoding can be zero-lag (causal), positive-lag (acausal), or even negative-lag. The latter would result in a very 'snappy' decoder that anticipated actions at the expense of decoding accuracy. The lag hyperparameter wouldn't need to be specified prior to training and could be freely adjusted at any time.

All datasets were curated to contain sorted spikes. Yet during online performance, decoding must typically rely on unsorted threshold crossings. Prior studies have found that moving from sorted spikes to threshold crossings produces a modest but tolerable drop in performance [76]. We similarly found that moving from sorted spikes to threshold crossings (selected to be from 'good' channels with reasonable signal-to-noise) produced only a small

drop in performance (Fig. S3d). The operations of MINT highlight why robustness is expected. When spikes from two neurons are combined, the resulting 'unit' acts statistically as if it were a high firing-rate neuron. Spiking is still approximately Poisson, because the sum of Poisson processes is a Poisson process. Thus, MINT's statistical assumptions remain appropriate.

Neural state estimation

MINT's success at behavioral decoding suggests underlying success in neural-state estimation. However, that would not necessarily have to be true. For example, the GRU also typically supported excellent behavioral decodes, but doesn't provide neural state estimation. MINT does (by its very construction) but whether those estimates are accurate must be determined directly.

To evaluate neural-state estimates, we submitted firing-rate predictions to the Neural Latents Benchmark [77], an online benchmark for latent variable models that compares acausal neural-state estimates across methods and datasets. The four primary datasets for the benchmark are MC_Maze, Area2_Bump, MC_RTT, and an additional dataset, DMFC_RSG, that contains neural recordings from dorsomedial frontal cortex while a monkey performed a cognitive timing task [78]. Three secondary datasets (MC_Maze-L, MC_Maze-M, and MC_Maze-S) contain additional sessions of the maze task with progressively fewer training trials (500, 250, 100, respectively).

By submitting to the Neural Latents Benchmark, we can ask how MINT performs relative to a set of 'baseline' methods. One baseline method (smoothed spikes) is traditional and simple. Other baseline methods, Gaussian Process Factor Analysis (GPFA) and Switching Linear Dynamical System (SLDS), are modern yet well-established. Finally, two baseline methods are quite new and cutting-edge: Neural Data Transformers (NDT) [40] and AutoLFADS [39]. These are expected to provide a high bar by which to judge the performance of other approaches.

Although the neural state is a well-defined quantity in spiking models [33], experimental data does not typically provide a ground-truth neural state against which estimates can be compared (unlike for behavioral-state decoding). Yet one can define attributes the neural state should possess, then determine whether estimates display those attributes. One critical attribute is prediction of spiking activity. This attribute is assessed via bits per spike, which is closely related to the log-likelihood of spiking observations given a set of firing rates (assuming Poisson spiking). MINT performed similarly to AutoLFADS and was consistently slightly better than NDT on the four primary datasets (Fig. 5a, *left*). MINT performed slightly better than AutoLFADS and NDT on the largest of the secondary datasets, and that advantage grew as dataset-size became smaller (Fig. 5a, *right*). MINT outperformed the other three baseline methods by modest-to-sizeable margins for all seven datasets.

MINT was not designed to maximize bits per spike – good performance in this regard simply results from the primary goal of estimating a neural state that is expressed in terms of firing rates. This matters because there exist spiking features – e.g. synchrony beyond that due to correlated rates – that some methods may be able to leverage. Doing so is reasonable but separate from MINT's goal of estimating a rate-based neural state. It is thus worth considering other attributes expected of good neural-state estimates. The Neural Latents Benchmark includes two such attributes.

First, good neural state estimates should be correct on average. Although ground-truth trial-averaged neural states are also unavailable, they can be approximated by PSTHs (trial-averaged smoothed spikes for each condition). Thus, trial-averaged neural state estimates should resemble PSTHs from the same condition, an attribute assessed by PSTH R^2. MINT achieved higher R^2 values than every other method on every dataset (Fig. 5b). This result relates to the fact that MINT estimates neural states near idealized trajectories 'built' from trial-averaged rates. Yet strong performance is not a trivial consequence of MINT's design; PSTH R^2 will be high only if condition (c), index (k), and the interpolation parameter (α) are accurately estimated.

Second, in sensorimotor areas during motor tasks, behavior should be decodable from the neural state. The benchmark thus measured R^2 for a linear decode of velocity from the neural state. MINT outperformed all baseline approaches (Fig. 5c) for all datasets. (MINT could of course have achieved even better decoding using its typical direct association, but that would have undermined the utility of this comparison).

In addition to the five baseline methods, the Neural Latents Benchmark solicited submissions from other methods and many were submitted. These submissions spanned a broad range in terms of performance (gray vertical lines in Fig. 5), highlighting the challenging nature of this problem. Some submissions involved ensemble approaches, some of which outperformed both MINT and the baseline methods in terms of bits per spike. For the primary datasets, the highest bits per spike overall was achieved by an ensemble of SpatioTemporal Neural Data Transformers [79], achieving .386 bits per spike on the MC_Maze dataset, compared to .330 for MINT. For the smaller secondary datasets, the very best submissions provided minimal-to-modest improvement over MINT in terms of bits per spike (Fig. 5a, *right*), and the majority performed less well.

The ability of some expressive neural networks (and some ensembles of these networks) to improve the bits per spike metric indicates there exists variability in the neural recordings that MINT does not capture. That variability could arise from behaviorally irrelevant fluctuations in the neural state, from synchrony based effects, or

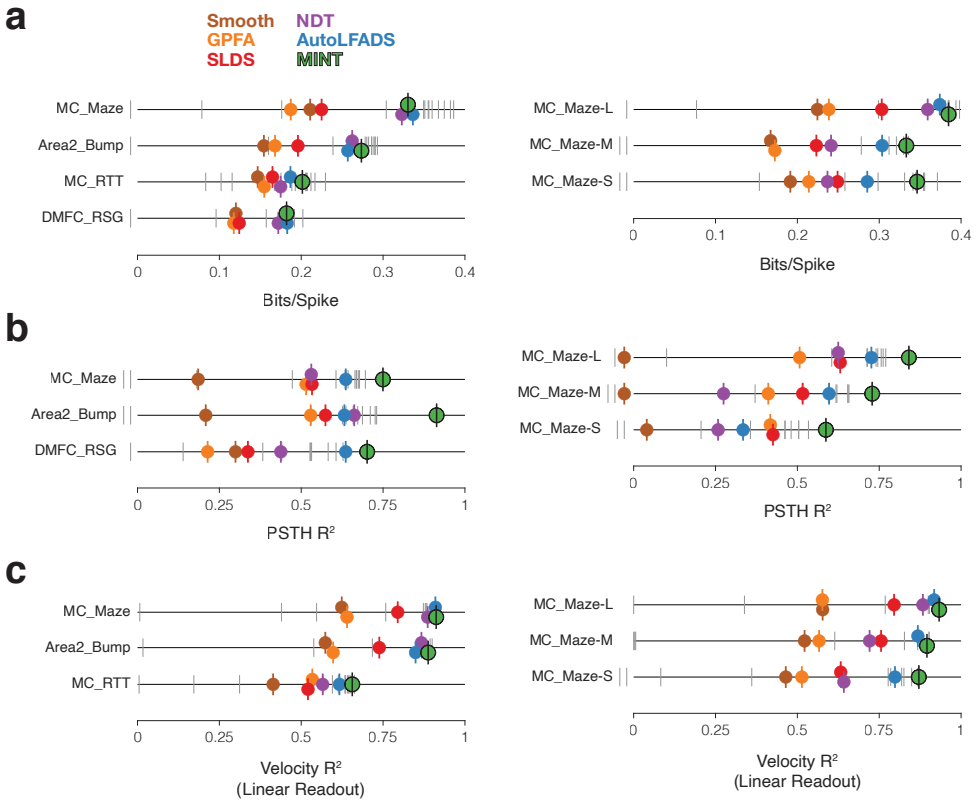

Figure 5. Evaluation of neural state estimates for seven datasets. a) Performance quantified using bits per spike. The benchmark's baseline methods have colored markers. All other submissions have gray markers. Vertical offsets and vertical ticks are used to increase visibility of data. Results with negative values are designated by markers to the left of zero, but their locations don't reflect the magnitude of the negative values. Data from Neural Latents Benchmark (`https://neurallatents.github.io/`). **b)** Performance quantified using PSTH R^2. **c)** Performance quantified using velocity R^2, after velocity was decoded linearly from the neural state estimate. A linear decode is used simply as a way of evaluating the quality of neural state estimates, especially in dimensions relevant to behavior.

from instabilities in the neural recordings. These are 'real' things and it is thus valid for a method to leverage them to predict spiking when that is the goal. Yet as noted above, they may often be incidental to one's goals in estimating the neural state, highlighting the need to consider additional attributes such as PSTH R^2 and velocity R^2. For PSTH R^2, MINT outperformed all other methods for all six datasets. For velocity R^2, MINT had the best performance for three out of six datasets and was very close to the highest R^2 values for the remaining three datasets (the largest deficit was ΔR^2 = .012) (Fig. 5c).

In summary, MINT's strong decode performance (Fig. 4) does indeed follow from good neural-state estimation. MINT provided estimates that were competitive with the best present methods. This includes existing well-documented methods, as well as a bevy of additional submissions. This implies that MINT could be used in situations where neural-state estimation is the goal, possibly including monitoring of the neural state for clinical purposes. MINT uses simple operations and is readily implemented causally – indeed this is typically how we would expect to see it used. Yet MINT's performance is typically similar to – and often better than – that of complex methods that would likely have to be limited to offline situations.

Practical considerations

Training & execution times

MINT is typically very fast to train (Table 1), on the order of seconds using generic hardware (no GPUs). Fast training is a consequence of the simple operations involved in constructing the library of neural-state trajectories: principally filtering of spikes and averaging across trials. Thus, MINT can be trained extremely quickly. At the same time we stress that MINT is a method for leveraging a trajectory library, not a method for constructing it. One may sometimes wish to use alternatives to trial-averaging, either of necessity or because they improve trajectory estimates. For example, for the MC_RTT task we used AutoLFADS to infer the library and training was consequently much slower (hours rather than seconds). Training time could be reduced back to seconds using a different approach – grouping

Data Set	Training Time (s)	Execution Time (ms)
Area2_Bump	4.8 ± 0.2	0.31 ± 0.03
MC_Cycle	20.3 ± 2.9	0.55 ± 0.03
MC_Maze	42.8 ± 0.6	2.26 ± 0.09
MC_Maze-L	10.8 ± 0.4	0.83 ± 0.02
MC_Maze-M	8.0 ± 0.4	0.80 ± 0.04
MC_Maze-S	5.8 ± 0.2	0.59 ± 0.07
MC_RTT	~ 3.2 hours	6.99 ± 0.28

Table 1. MINT training times and average execution times (average time it took to decode a 20 ms bin of spiking observations). To be appropriate for real-time applications, execution times must be shorter than the bin width. Note that the 20 ms bin width does not prevent MINT from decoding every millisecond; MINT updates the inferred neural state and associated behavioral state between bins, using neural and behavioral trajectories that are sampled every millisecond. For all table entries (except MC_RTT training time), means and standard deviations were computed across 10 train/test runs for each dataset. Training times exclude loading datasets into memory and any hyperparameter optimization. Timing measurements taken on a Macbook Pro (on CPU) with 32GB RAM and a 2.3 GHz 8-Core Intel Core i9 processor. For MC_RTT, training involved running AutoLFADS twice (and averaging the resulting rates) to generate neural trajectories. This training procedure utilized 10 GPUs and took ∼1.6 hours per run. For AutoLFADS, hyperparameter optimization and model fitting procedures are intertwined. Thus, the training time reported includes time spent optimizing hyperparameters.

into pseudo-conditions – but performance was then reduced by a modest but meaningful amount. Thus, training will typically be very fast, but one may choose time-consuming methods when appropriate.

Execution times were well below the threshold needed for real-time applications (Table 1), even using a laptop computer. State estimation in MINT is $\mathcal{O}(NC)$ when making the simplifying assumption that all conditions are of equal length. Thus, the computational burden grows only linearly with neurons or with conditions. Additionally, much of the estimation procedure in MINT is condition-specific and therefore parallelizable. This makes MINT a good candidate for decoding in BCI applications that involve large neuron counts and/or large behavioral repertoires.

Performance on small training sets

To investigate robustness to training with small trial-counts, we leveraged the fact that the four maze-reaching datasets – one primary (MC_Maze), and three secondary (MC_Maze-L, MC_Maze-M, and MC_Maze-S) – contained training sets of decreasing size. For comparison, we chose the GRU and the feedforward neural network because of their generally good position- and velocity-decoding performance (Fig. 4).

On the largest training set, MINT and the GRU achieved similar performance for position decoding (Fig. 6a, R^2 = .963 and R^2 = .962, respectively). The performance gap widened as training sets became smaller (R^2 = .926 versus R^2 = .880 for the smallest dataset). Similar results were found for velocity decoding: the performance gap between MINT and the GRU widened for smaller training sets. Thus, MINT was quite robust to small training sets: an approximately five-fold reduction in training data led to a decline in R^2 of only .037 (position) and .050 (velocity). The feedforward network did not show as consistent a performance rolloff with number of training trials, but also underperformed overall.

Recording instabilities

With respect to recording quality, one desires two types of robustness. First, allowing for retraining, decoding should be robust to the across-session loss of neurons. To simulate such loss, we performed a neuron dropping analysis on the MC_Maze-L dataset (Fig. 6b, *solid lines*). Training was repeated after neurons were dropped. Average R^2 for position decoding remained above 95% of peak performance until fewer than 58 neurons (of 162) remained. Results were similar for velocity decoding: performance was >95% of its peak until fewer than 62 neurons remained. We repeated this analysis for the Wiener filter. Despite its overall lower performance, the Wiener filter provides a useful comparison because (like MINT) it is easy to retrain quickly, with reliable results, across the many differently sized training sets examined here. The Wiener filter was also fairly robust, but less so than MINT. To remain above 95% of peak performance, the Wiener filter needed approximately twice as many remaining neurons: 118 (Wiener filter) versus 58 (MINT) for position and 126 (Wiener filter) versus 62 (MINT) for velocity.

Second, because there can be occasional within-session loss of isolations or individual channels, one desires robustness to modest neuron loss even without retraining. We simulated within-session neuron loss by eliminating all spikes from a subset of neurons (Fig. 6b, *dashed lines*) with no retraining. MINT's performance was robust to modest neuron loss. Losing fewer than 20 neurons caused essentially no performance loss. Position and velocity decoding remained above 95% of peak performance until fewer than 113 and 109 neurons remained, respectively.

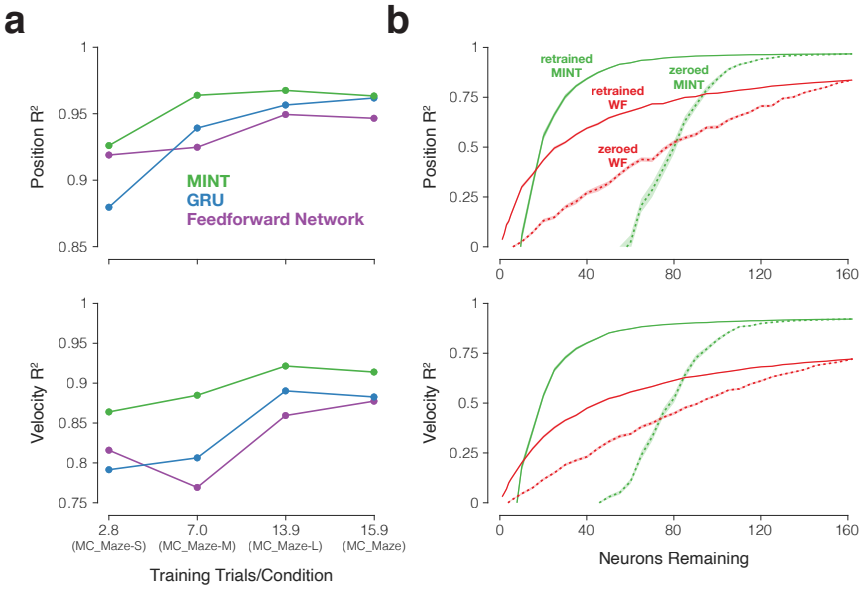

Figure 6. Decoding robustness in the face of small training sets and neuron counts. a) R^2 values for MINT and two neural network decoders (GRU and the feedforward network) when decoding position and velocity from four maze datasets with progressively fewer training trials per condition. **b)** R^2 values for MINT and the Wiener filter when decoding position and velocity from MC_Maze-L with progressively fewer neurons. 'Retrained' results (*solid lines*) show mean performance (across 50 random drops at each neuron count) when the decoders are trained and tested on reduced neuron sets. 'Zeroed' results (*dashed lines*) show mean performance when the decoders are trained on all 162 neurons and, during testing, the 'dropped' neurons' spike counts are set to zero without retraining the decoder. Shaded regions depict standard errors across repeated droppings with different neurons.

In contrast, the Wiener filter had a more linear performance roll off.

Thus, MINT is robust to modest neuron-loss without retraining. At the same time, we stress that 'retraining' MINT to account for neuron loss is incredibly simple – it simply involves ignoring that neuron when computing spike likelihoods. Unlike nearly all other methods (including the Wiener filter) the loss of one neuron does not require changing parameters that apply to other neurons. Thus, if neuron loss is detected, retraining can occur on the fly with no need to pause decoding.

Interpretability

One form of interpretability relates to whether an algorithm uses 'black box' optimization to learn a solution. Neural networks can be highly expressive and learn constraints directly from data. This approach is useful when appropriate constraints are unclear. However, when constraints are known, one can rely more on model assumptions and less on expressivity, yielding robust methods that perform accurately with less training data. MINT is an interpretable algorithm that relies on strong assumptions.

The performance of interpretable methods can reflect the extent to which their assumptions match the statistics of the data. Consider the Kalman filter – an interpretable algorithm – and its performance on the MC_Maze reaching dataset. The Kalman filter produces a relatively poor decode (R^2 = .609 for hand velocity). This doesn't imply the Kalman filter is a poor algorithm, simply that its assumptions may not be well-aligned with the data. We confirmed this by generating synthetic data whose properties are closer to those assumed by the Kalman filter. The Kalman filter then performed quite competitively (Fig. S4). While expected, this result underscores an important point. MINT does not outperform the Kalman filter because it is intrinsically a better method. Instead, MINT performs well because its assumptions, guided by an understanding of neural geometry, are presumably a good match to the underlying properties of the data.

MINT is also interpretable regarding why each prediction is made. Suppose that, at a given moment during decoding, MINT selects neural state $\overline{\mathbf{x}}_k^c$ as the most likely state, and thus renders its decode using the associated behavioral state $\overline{\mathbf{z}}_k^c$. Selection of $\overline{\mathbf{x}}_k^c$ comes with an associated likelihood that conveys information regarding how well the spiking data matched that neural state. Likelihoods are known not only for this 'selected' state, but for all non-selected states on all trajectories in the library. These likelihoods can be analyzed to understand why the selected state was chosen over the others, how close the algorithm was to selecting a different state, and which neurons contributed most strongly to state selection. Because of its probability-based approach, MINT is readily

modifiable and extensible in a variety of ways (Fig. S5).

Discussion

Implications regarding neural geometry

All neural decode algorithms and state estimators make assumptions regarding the structure of neural activity and/or its relationship to behavior. Correctly specifying these assumptions can yield accurate, interpretable methods. MINT is the outcome of our endeavor to codify observations regarding motor-cortex population geometry into statistical assumptions for a decoder. Some observations are recent and some are documented here for the first time.

Low trajectory tangling [63] creates sparsely distributed trajectories whose geometry differs from behavioral trajectories, both muscle and kinematic. In particular, distance is poorly preserved between neural and behavioral trajectories. Neighboring states in behavioral space can be quite distant in neural space. These observations challenge two common assumptions regarding motor cortex activity. The first is that there exist strong isometries between neural activity and to-be-decoded parameters. Traditional tuning-centric hypotheses embody this assumption. Yet even as the field has moved towards factor-dynamics based hypotheses, it has remained reasonable to assume strong isometries in a subset of high-variance dimensions. Yet as the model spiking network in Figure 3b illustrates, it is also likely that motor-cortex outputs are encoded in low-variance dimensions. If so, isometry-based decoding becomes limited to low-variance dimensions, or to isometries that are incidental and task-specific [20]. MINT avoids the need to make assumptions regarding isometries by using a direct decode.

The second common assumption is that the range of observable neural states can be approximately described by a covariance matrix, and/or by the associated low-dimensional subspace obtained from a subset of its eigenvectors (the principal components). There has always been some tension associated with this assumption. Dimensionality reduction has been – and almost certainly will continue to be – a powerful analysis tool. Competing hypotheses often make contrasting predictions regarding what occurs in high-variance dimensions. Dimensionality reduction methods are ideally suited for testing those predictions. At the same time, it is unclear when and whether motor-cortex trajectories are themselves low-dimensional. Even in studies that explicitly consider an occupied subspace, that subspace is high-dimensional relative to traditional assumptions [80], and learning can 'open up' new dimensions [81]. Studies have also stressed the possibility of a nonlinear manifold embedded in a higher-dimensional subspace [82–85]. Multiple studies have stressed the remarkably high-dimensional nature of motor cortex activity [66, 67, 86, 87]. Network models of motor cortex activity [88, 89] have a high ratio of across-neuron dimensionality to across-condition dimensionality [86]. Networks that produce extended temporally structured outputs can leverage trajectories whose geometry resembles complex and well-separated 'tubes' embedded in a very high-dimensional space [90].

Present results – both analysis of data and the successful decode provided by MINT – support the idea that 'subspace dimensionality' may not be the best way to describe the constraints governing neural trajectories. Low trajectory tangling can, depending on the circumstances, be compatible with either low or high dimensionality. What is more likely to be consistent are properties of the not-easily-described nonlinear manifold, where large 'voids' (e.g. the middle of circular trajectories) remain unoccupied [63, 64]. Precisely because this geometry is difficult to describe with a model, MINT approximates the manifold using the empirical trajectories as a 'scaffolding', with interpolation producing a mesh of states that are considered during decoding.

Embracing these aspects of trajectory geometry produced excellent performance: MINT accurately estimated the neural state and decoded a broad range of behavioral variables across a variety of tasks and brain areas. Good decode performance, on its own, is insufficient to validate the assumptions of a decode method and we stress caution in that regard. This is particularly true because many of the scientific findings that inspired MINT are not, strictly speaking, assumptions of the method itself. MINT is designed to work well when trajectories are sparsely distributed, with low tangling, and spread out in a high-dimensional space. Yet MINT can perform well in the absence of these features. For example, MINT performed well on simulated data that was instead well-matched to the assumptions of a Kalman filter (MINT and the Kalman filter performed comparably well in that case).

Thus, good decode performance does not necessarily imply MINT's assumptions are uniquely correct. Nevertheless, comparisons can aid interpretation. For the empirical data, MINT consistently outperformed the Kalman filter, often by a wide margin. This indicates that the data contain properties that MINT, but not the Kalman filter, is poised to leverage. Given the analysis in Figure 1, some of the most likely properties are those that motivated MINT's design: sparsely distributed low-tangled trajectories. MINT's assumption of Poisson spiking statistics may also have contributed, yet MINT had a much smaller advantage for simulated data where spiking was Poisson but neural geometry matched the Kalman filter's assumptions Fig. S4. Thus, it is likely that MINT gains much of its advantage via constraints on the neural-state geometry that are more accurate than those imposed by the Kalman filter. Comparison with expressive machine-learning approaches makes a similar point from the other direction. Highly expressive methods, through training, should be able to learn and leverage a range of neural constraints.

Given that MINT was competitive with these methods, its assumptions are likely to be a good match to the statistics of the data.

Generalization

MINT readily performs certain types of generalization. Interpolation allows MINT to generalize spatially, to neural states (and corresponding behavioral states) that are between trajectories observed during training. MINT can also generalize temporally, by selecting mostly-likely states that move faster (or slower) in neural state space than those present during training. MINT can also generalize compositionally: switching between trajectories or exiting them early. As a simple example, a library consisting of a single 4-cycle trajectory from the cycling task would allow MINT to decode a 2-cycle movement or a 200-cycle movement. Thus, MINT is not stuck on the library trajectories, nor is it stuck traversing trajectories at fixed speeds or for fixed durations.

Yet while MINT can follow trajectories through the mesh that are quite different from those in the library, all decoded states will be on the mesh. MINT is not capable of generalizing to trajectories that traverse regions of state space far from the mesh. This is relevant, because empirical evidence indicates that 'new' behaviors can sometimes be associated with neural trajectories that are very different from all previously observed trajectories – often occupying orthogonal subspaces [63, 91–93]. MINT could be modified to allow extrapolation outside the library, but we suspect that this form of generalization will still not typically be available. As illustrated in Figure S2, it will be extremely difficult for any decoder to successfully decode neural trajectories that are orthogonal to those in the training set. For example, none of the methods we evaluated were able to generalize from forward to backward cycling, which were associated with nearly orthogonal neural trajectories. However, MINT's computation of log-likelihoods does lend it a potential advantage in handling novel or unexpected behaviors. Using the training set, one can roughly estimate the distribution of log-likelihoods. If, during decoding, log-likelihoods become vastly lower than expected, that suggests decoding is struggling with a movement too far from the training set. This knowledge would give the technician options, including decoding zero movement (on the grounds that no movement is better than a large error) or pausing to retrain. The latter could leverage another advantage of MINT: adding a trajectory to the library – and thus expanding the mesh – is simple and no other retraining or modification of parameters is needed.

Multi-task decoding

Across-task generalization may often be limited by the above considerations. Yet this does not imply that MINT cannot decode in a multi-task regime. On the contrary, MINT is ideally suited to decoding from a behavioral repertoire spanning two or more tasks – each task simply needs to be represented in the mesh with an appropriate set of trajectories. And as noted above, trajectories corresponding to a new task can be easily added to the library.

MINT's facility with multi-task decoding is particularly relevant because, historically, decoding approaches have had to vary across tasks. For example, linear decoding of velocity works well during reaching but not cycling [20]. Nevertheless, Schroeder et al. [20] were able to achieve accurate closed-loop decoding of cycling by nonlinearly leveraging statistical regularities in a task-specific neural subspace. MINT embraces both task-specificity and unification. Each task contributes a unique set of trajectories to the mesh. A unified decoding strategy then applies both within and across tasks. This avoids the need to engineer bespoke decoders for each task individually and facilitates multi-task decoding that doesn't rely on an additional meta-decoder to identify the appropriate task-specific decoder at each moment.

Future avenues

MINT's probability-based framework is amenable to principled extensions. Rather than simply decode the state with the highest data likelihood, one could incorporate priors. Priors could be fixed or guided by incoming information (e.g. an eye tracker). Decoding could focus on utility functions rather than probabilities; a patient may value avoiding certain types of errors more than others. For example, erroneous transitions from stationary to moving may be jarring or unsafe, and could be dissuaded via an appropriately designed utility function.

As noted above, MINT is amenable to a variety of methods – simple or sophisticated – for inferring the trajectory library. Possible future approaches including leveraging the typical neural geometry for a particular task to allow a patient-specific mesh to be learned using fewer trials. This might even be done in an unsupervised way: a trajectory library could be built without behavioral labels, then appropriately linked to a 'reference' behavioral library via knowledge of task-specific neural geometry. As a simple example, cycling across different speeds produces a tube-shaped manifold with different velocities at each end [64]. An empirically observed tube might be given behavioral labels based on this knowledge. These observations may be relevant to a primary goal of MINT: potential use as an online decoder for real-time closed-loop performance. In situations involving paralyzed patients, training typically involves asking subjects to attempt to move, which creates uncertainty regarding the exact behavioral labels

for each moment. The above strategies may aid in this regard. The approach of Gilja et al. [13] would also likely be helpful. They found that re-fitting Kalman filter parameters online improved closed-loop performance. One could similarly use an initial mesh to allow decoding to gain traction, then 're-mesh' MINT online to improve performance.

Acknowledgments

We thank Felix Pei, Chethan Pandarinath, and the rest of the Neural Latents Benchmark team for curating many of the datasets used in this paper and releasing them publicly. We thank those who originally collected those datasets: Raeed Chowdhury (Area2_Bump), Hansem Sohn (DMFC_RSG), Matt Kaufman and Mark Churchland (MC_Maze, MC_Maze-L,M,S), and Joseph O'Doherty (MC_RTT). We additionally thank Felix and Chethan for providing AutoLFADS rates for the MC_RTT dataset. We thank Karen Schroeder for collecting and preprocessing the MC_Cycle dataset in collaboration with the first and last authors. We thank Yana Pavlova for expert animal care while collecting the MC_Cycle dataset. We thank Brian DePasquale for providing code for simulating the artificial multitask spiking network. We thank Andrew Zimnik, Elom Amematsro, and Eric Trautmann for helping to present this work at NCM 2022 when the first author had COVID. This work was supported by the Simons Foundation and the Grossman Center for the Statistics of Mind.

Author contributions

SP and MC designed the decoder. JC provided input on how to conceptualize and formalize the decoder. SP wrote the software. SP and MC designed data analyses. SP performed analyses. QW provided guidance regarding analyses. SP and MC wrote the paper. All authors contributed to editing.

Competing interests

SP, JC, and MC hold a patent pertaining to this work. The patent has been licensed to Blackrock Neurotech. The authors declare no additional competing interests.

Code availability

- MINT is available at `https://github.com/seanmperkins/mint` (MATLAB).
- Other decoders are available at `https://github.com/seanmperkins/bci-decoders` (Python).
- Multitask spiking network was simulated with code from DePasquale et al. [33] that is available at `https://github.com/briandepasquale/factor-based-spiking-nets/` (MATLAB).

Data availability

All empirical datasets (except MC_Cycle) were curated by the Neural Latents Benchmark team [77] and made publicly available on DANDI (linked below). The MC_Cycle dataset is available upon request. Useful functions for loading the DANDI datasets are available at `https://github.com/neurallatents/nlb_tools`.

- Area2_Bump: `https://dandiarchive.org/dandiset/000127`
- DMFC_RSG: `https://dandiarchive.org/dandiset/000130`
- MC_Maze: `https://dandiarchive.org/dandiset/000128`
- MC_Maze-L: `https://dandiarchive.org/dandiset/000138`
- MC_Maze-M: `https://dandiarchive.org/dandiset/000139`
- MC_Maze-S: `https://dandiarchive.org/dandiset/000140`
- MC_RTT: `https://dandiarchive.org/dandiset/000129`

References

[1] Jose M. Carmena et al. "Learning to Control a Brain–Machine Interface for Reaching and Grasping by Primates". In: *PLoS Biology* 1.2 (2003), e42. ISSN: 1544-9173. DOI: 10.1371/journal.pbio.0000042.

[2] Leigh R. Hochberg et al. "Neuronal ensemble control of prosthetic devices by a human with tetraplegia". In: *Nature* 442.7099 (2006), pp. 164–171. ISSN: 0028-0836. DOI: 10.1038/nature04970.

[3] Meel Velliste et al. "Cortical control of a prosthetic arm for self-feeding". In: *Nature* 453.7198 (2008), pp. 1098–1101. ISSN: 0028-0836. DOI: 10.1038/nature06996.

[4] Leigh R. Hochberg et al. "Reach and grasp by people with tetraplegia using a neurally controlled robotic arm". In: *Nature* 485.7398 (2012), pp. 372–375. ISSN: 0028-0836. DOI: 10.1038/nature11076.

[5] Jennifer L. Collinger et al. "High-performance neuroprosthetic control by an individual with tetraplegia". In: *The Lancet* 381.9866 (2013), pp. 557–564. ISSN: 0140-6736. DOI: 10.1016/s0140-6736(12)61816-9.

[6] B Wodlinger et al. "Ten-dimensional anthropomorphic arm control in a human brain-machine interface: difficulties, solutions, and limitations". In: *Journal of Neural Engineering* 12.1 (2015), p. 016011. ISSN: 1741-2552. DOI: 10.1088/1741-2560/12/1/016011.

[7] Chet T. Moritz, Steve I. Perlmutter, and Eberhard E. Fetz. "Direct control of paralysed muscles by cortical neurons". In: *Nature* 456.7222 (2008), pp. 639–642. ISSN: 0028-0836. DOI: 10.1038/nature07418.

[8] C. Ethier et al. "Restoration of grasp following paralysis through brain-controlled stimulation of muscles". In: *Nature* 485.7398 (2012), pp. 368–371. ISSN: 0028-0836. DOI: 10.1038/nature10987.

[9] Chad E. Bouton et al. "Restoring cortical control of functional movement in a human with quadriplegia". In: *Nature* 533.7602 (2016), pp. 247–250. ISSN: 0028-0836. DOI: 10.1038/nature17435.

[10] A. Bolu Ajiboye et al. "Restoration of reaching and grasping movements through brain-controlled muscle stimulation in a person with tetraplegia: a proof-of-concept demonstration". In: *The Lancet* 389.10081 (2017), pp. 1821–1830. ISSN: 0140-6736. DOI: 10.1016/s0140-6736(17)30601-3.

[11] Dawn M. Taylor, Stephen I. Helms Tillery, and Andrew B. Schwartz. "Direct Cortical Control of 3D Neuroprosthetic Devices". In: *Science* (2002).

[12] Mijail D. Serruya et al. "Instant neural control of a movement signal". In: *Nature* 416.6877 (2002), pp. 141–142. ISSN: 0028-0836. DOI: 10.1038/416141a.

[13] Vikash Gilja et al. "A high-performance neural prosthesis enabled by control algorithm design". In: *Nature Neuroscience* 15.12 (2012), pp. 1752–1757. ISSN: 1097-6256. DOI: 10.1038/nn.3265.

[14] Beata Jarosiewicz et al. "Virtual typing by people with tetraplegia using a self-calibrating intracortical brain-computer interface". In: *Science Translational Medicine* 7.313 (2015), 313ra179. ISSN: 1946-6234. DOI: 10.1126/scitranslmed.aac7328.

[15] Paul Nuyujukian et al. "A High-Performance Keyboard Neural Prosthesis Enabled by Task Optimization". In: *IEEE Transactions on Biomedical Engineering* 62.1 (2015), pp. 21–29. ISSN: 0018-9294. DOI: 10.1109/tbme.2014.2354697.

[16] Maryam M. Shanechi et al. "Rapid control and feedback rates enhance neuroprosthetic control". In: *Nature Communications* 8.1 (2017), p. 13825. DOI: 10.1038/ncomms13825.

[17] Chethan Pandarinath et al. "High performance communication by people with paralysis using an intracortical brain-computer interface". In: *eLife* 6 (2017), e18554. DOI: 10.7554/elife.18554.

[18] Camilo Libedinsky et al. "Independent Mobility Achieved through a Wireless Brain-Machine Interface". In: *PLOS ONE* 11.11 (2016), e0165773. DOI: 10.1371/journal.pone.0165773.

[19] Sankaranarayani Rajangam et al. "Wireless Cortical Brain-Machine Interface for Whole-Body Navigation in Primates". In: *Scientific Reports* 6.1 (2016), p. 22170. DOI: 10.1038/srep22170.

[20] Karen E Schroeder et al. "Cortical Control of Virtual Self-Motion Using Task-Specific Subspaces". In: *The Journal of Neuroscience* 42.2 (2022), pp. 220–239. ISSN: 0270-6474. DOI: 10.1523/jneurosci.2687-20.2021.

[21] Gopala K. Anumanchipalli, Josh Chartier, and Edward F. Chang. "Speech synthesis from neural decoding of spoken sentences". In: *Nature* 568.7753 (2019), pp. 493–498. ISSN: 0028-0836. DOI: 10.1038/s41586-019-1119-1.

[22] Guy H. Wilson et al. "Decoding spoken English from intracortical electrode arrays in dorsal precentral gyrus". In: *Journal of Neural Engineering* 17.6 (2020), p. 066007. ISSN: 1741-2560. DOI: 10.1088/1741-2552/abbfef.

[23] Francis Willett et al. "A high-performance speech neuroprosthesis". In: *bioRxiv* (2023), p. 2023.01.21.524489. DOI: 10.1101/2023.01.21.524489.

[24] Francis R. Willett et al. "High-performance brain-to-text communication via handwriting". In: *Nature* 593.7858 (2021), pp. 249–254. ISSN: 0028-0836. DOI: 10.1038/s41586-021-03506-2.

[25] S. Musallam et al. "Cognitive Control Signals for Neural Prosthetics". In: *Science* 305.5681 (2004), pp. 258–262. ISSN: 0036-8075. DOI: 10.1126/science.1097938.

[26] Joni D. Wallis. "Decoding Cognitive Processes from Neural Ensembles". In: *Trends in Cognitive Sciences* 22.12 (2018), pp. 1091–1102. ISSN: 1364-6613. DOI: 10.1016/j.tics.2018.09.002.

[27] Omid G. Sani et al. "Mood variations decoded from multi-site intracranial human brain activity". In: *Nature Biotechnology* 36.10 (2018), pp. 954–961. ISSN: 1087-0156. DOI: 10.1038/nbt.4200.

[28] Ali Yousefi et al. "Decoding Hidden Cognitive States From Behavior and Physiology Using a Bayesian Approach". In: *Neural Computation* 31.9 (2019), pp. 1751–1788. ISSN: 0899-7667. DOI: 10.1162/neco_a_01196.

[29] Nicole R. Provenza et al. "Decoding task engagement from distributed network electrophysiology in humans". In: *Journal of Neural Engineering* 16.5 (2019), p. 056015. ISSN: 1741-2552. DOI: 10.1088/1741-2552/ab2c58.

[30] Aswin Chari et al. "Microelectrode recordings in human epilepsy: A case for clinical translation?" In: *Brain Communications* 2.2 (2020), fcaa082. ISSN: 2632-1297. DOI: 10.1093/braincomms/fcaa082.

[31] Benjamin R. Cowley et al. "DataHigh: graphical user interface for visualizing and interacting with high-dimensional neural activity". In: *Journal of Neural Engineering* 10.6 (2013), p. 066012. ISSN: 1741-2552. DOI: 10.1088/1741-2560/10/6/066012.

[32] Diogo Peixoto et al. "Decoding and perturbing decision states in real time". In: *Nature* 591.7851 (2021), pp. 604–609. ISSN: 0028-0836. DOI: 10.1038/s41586-020-03181-9.

[33] Brian DePasquale et al. "The centrality of population-level factors to network computation is demonstrated by a versatile approach for training spiking networks". In: *Neuron* (2023). ISSN: 0896-6273. DOI: 10.1016/j.neuron.2022.12.007.

[34] Byron M. Yu et al. "Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity". In: vol. 21. 2009.

[35] Jakob H. Macke et al. "Empirical models of spiking in neural populations". In: vol. 24. 2012.

[36] Yuanjun Gao et al. "High-dimensional neural spike train analysis with generalized count linear dynamical systems". In: vol. 28. 2015.

[37] Jonathan C. Kao et al. "Single-trial dynamics of motor cortex and their applications to brain-machine interfaces". In: *Nature Communications* 6.1 (2015), p. 7759. DOI: 10.1038/ncomms8759.

[38] Chethan Pandarinath et al. "Inferring single-trial neural population dynamics using sequential auto-encoders". In: *Nature Methods* 15.10 (2018), pp. 805–815. ISSN: 1548-7091. DOI: 10.1038/s41592-018-0109-9.

[39] Mohammad Reza Keshtkaran et al. "A large-scale neural network training framework for generalized estimation of single-trial population dynamics". In: *Nature Methods* (2022), pp. 1–6. ISSN: 1548-7091. DOI: 10.1038/s41592-022-01675-0.

[40] Joel Ye and Chethan Pandarinath. "Representation learning for neural population activity with Neural Data Transformers". In: *Neurons, Behavior, Data Analysis, and Theory* (2021). DOI: 10.51628/001c.27358.

[41] Caleb T. Kemere et al. "Decoding of Plan and Peri-Movement Neural Signals in Prosthetic Systems". In: *IEEE Workshop on Signal Processing Systems* (2002), pp. 276–283. DOI: 10.1109/sips.2002.1049722.

[42] Jonathan C. Kao et al. "A High-Performance Neural Prosthesis Incorporating Discrete State Selection With Hidden Markov Models". In: *IEEE Transactions on Biomedical Engineering* 64.4 (2017), pp. 935–945. ISSN: 0018-9294. DOI: 10.1109/tbme.2016.2582691.

[43] W. Wu et al. "Neural Decoding of Cursor Motion Using a Kalman Filter". In: vol. 15. 2003.

[44] A. E. Brockwell, A. L. Rojas, and R. E. Kass. "Recursive Bayesian Decoding of Motor Cortical Signals by Particle Filtering". In: *Journal of Neurophysiology* 91.4 (2004), pp. 1899–1907. ISSN: 0022-3077. DOI: 10.1152/jn.00438.2003.

[45] Caleb Kemere, Maneesh Sahanil, and Teresa Meng. "Robust Neural Decoding of Reaching Movements for Prosthetic Systems". In: *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE Cat. No. 03CH37439)* 3 (2003), pp. 2079–2082. DOI: 10.1109/iembs.2003.1280146.

[46] C. Kemere, K. V. Shenoy, and T. H. Meng. "Model-based neural decoding of reaching movements: a maximum likelihood approach". In: *IEEE Transactions on Biomedical Engineering* 51.6 (2004), pp. 925–932. ISSN: 0018-9294. DOI: 10.1109/tbme.2004.826675.

[47] Caleb Kemere et al. "Model-Based Decoding of Reaching Movements for Prosthetic Systems". In: *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society* 2 (2004), pp. 4524–4528. ISSN: 1557-170X. DOI: 10.1109/iembs.2004.1404256.

[48] Caleb Kemere et al. "Detecting Neural-State Transitions Using Hidden Markov Models for Motor Cortical Prostheses". In: *Journal of Neurophysiology* 100.4 (2008), pp. 2441–2452. ISSN: 0022-3077. DOI: 10.1152/jn.00924.2007.

[49] Byron M. Yu et al. "Mixture of Trajectory Models for Neural Decoding of Goal-Directed Movements". In: *Journal of Neurophysiology* 97.5 (2007), pp. 3763–3780. ISSN: 0022-3077. DOI: 10.1152/jn.00482.2006.

[50] Omid G. Sani et al. "Modeling behaviorally relevant neural dynamics enabled by preferential subspace identification". In: *Nature Neuroscience* 24.1 (2021), pp. 140–149. ISSN: 1097-6256. DOI: 10.1038/s41593-020-00733-0.

[51] Eric A. Pohlmeyer et al. "Prediction of upper limb muscle activity from motor cortical discharge during reaching". In: *Journal of Neural Engineering* 4.4 (2007), p. 369. ISSN: 1741-2552. DOI: 10.1088/1741-2560/4/4/003.

[52] Joshua I. Glaser et al. "Machine learning for neural decoding". In: *eNeuro* 7.4 (2020), ENEURO.0506–19.2020. ISSN: 2373-2822. DOI: 10.1523/eneuro.0506-19.2020.

[53] Nur Ahmadi, Timothy G. Constandinou, and Christos-Savvas Bouganis. "Robust and accurate decoding of hand kinematics from entire spiking activity using deep learning". In: *Journal of Neural Engineering* 18.2 (2021), p. 026011. ISSN: 1741-2560. DOI: 10.1088/1741-2552/abde8a.

[54] Steffen Schneider, Jin Hwa Lee, and Mackenzie Weygandt Mathis. "Learnable latent embeddings for joint behavioral and neural analysis". In: *arXiv* (2022). DOI: 10.48550/arxiv.2204.00673. eprint: 2204.00673.

[55] Zheng Li et al. "Unscented Kalman Filter for Brain-Machine Interfaces". In: *PLoS ONE* 4.7 (2009), e6243. DOI: 10.1371/journal.pone.0006243.

[56] David Sussillo et al. "A recurrent neural network for closed-loop intracortical brain–machine interface decoders". In: *Journal of Neural Engineering* 9.2 (2012), p. 026027. ISSN: 1741-2552. DOI: 10.1088/1741-2560/9/2/026027.

[57] David Sussillo et al. "Making brain–machine interfaces robust to future neural variability". In: *Nature Communications* 7.1 (2016), p. 13749. DOI: 10.1038/ncomms13749. eprint: 1610.05872.

[58] Joseph G. Makin et al. "Superior arm-movement decoding from cortex with a new, unsupervised-learning algorithm". In: *Journal of Neural Engineering* 15.2 (2018), p. 026010. ISSN: 1741-2552. DOI: 10.1088/1741-2552/aa9e95.

[59] Michael A. Schwemmer et al. "Meeting brain–computer interface user performance expectations using a deep neural network decoding framework". In: *Nature Medicine* 24.11 (2018), pp. 1669–1676. ISSN: 1078-8956. DOI: 10.1038/s41591-018-0171-y.

[60] Po-He Tseng et al. "Decoding Movements from Cortical Ensemble Activity Using a Long Short-Term Memory Recurrent Network". In: *Neural Computation* 31.6 (2019), pp. 1085–1113. ISSN: 0899-7667. DOI: 10.1162/neco_a_01189.

[61] Eberhard E. Fetz. "Are movement parameters recognizably coded in the activity of single neurons?" In: *Behavioral and Brain Sciences* 15.4 (1992), pp. 679–690.

[62] Mark M Churchland et al. "Neural population dynamics during reaching". In: *Nature* 487.7405 (2012), pp. 51–56. ISSN: 0028-0836. DOI: 10.1038/nature11129.

[63] Abigail A. Russo et al. "Motor Cortex Embeds Muscle-like Commands in an Untangled Population Response". In: *Neuron* 97.4 (2018), 953–966.e8. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2018.01.004.

[64] Shreya Saxena et al. "Motor cortex activity across movement speeds is predicted by network-level strategies for generating muscle activity". In: *eLife* 11 (2022), e67620. DOI: 10.7554/elife.67620.sa0.

[65] Matthew T. Kaufman et al. "The Largest Response Component in the Motor Cortex Reflects Movement Timing but Not Movement Type". In: *eNeuro* 3.4 (2016), ENEURO.0085–16.2016. DOI: 10.1523/eneuro.0085-16.2016.

[66] Mark M. Churchland and Krishna V. Shenoy. "Temporal Complexity and Heterogeneity of Single-Neuron Activity in Premotor and Motor Cortex". In: *Journal of Neurophysiology* 97.6 (2007), pp. 4235–4257. ISSN: 0022-3077. DOI: 10.1152/jn.00095.2007.

[67] Najja J. Marshall et al. "Flexible neural control of motor units". In: *Nature Neuroscience* (2022), pp. 1–13. ISSN: 1097-6256. DOI: 10.1038/s41593-022-01165-8.

[68] Connor Brennan et al. "One dimensional approximations of neuronal dynamics reveal computational strategy". In: *PLOS Computational Biology* 19.1 (2023), e1010784. ISSN: 1553-734X. DOI: 10.1371/journal.pcbi.1010784.

[69] Matthew D. Golub et al. "Learning by neural reassociation". In: *Nature Neuroscience* 21.4 (2018), pp. 607–616. ISSN: 1097-6256. DOI: 10.1038/s41593-018-0095-3.

[70] Alan Degenhart et al. "Constraints on the time course of neural population activity". In: *Cosyne Abstracts* III-59 (2020).

[71] Vivek R. Athalye et al. "The brain uses invariant dynamics to generalize outputs across movements". In: *bioRxiv* (2021), p. 2021.08.27.457931. DOI: 10.1101/2021.08.27.457931.

[72] Mark M. Churchland et al. "Cortical Preparatory Activity: Representation of Movement or First Cog in a Dynamical Machine?" In: *Neuron* 68.3 (2010). ISSN: 0896-6273. DOI: 10.1016/j.neuron.2010.09.015.

[73] Raeed H Chowdhury, Joshua I Glaser, and Lee E Miller. "Area 2 of primary somatosensory cortex encodes kinematics of the whole arm". In: *eLife* 9 (2020), e48198. DOI: 10.7554/elife.48198.

[74] Johan Wessberg et al. "Real-time prediction of hand trajectory by ensembles of cortical neurons in primates". In: *Nature* 408.6810 (2000), pp. 361–365. ISSN: 0028-0836. DOI: 10.1038/35042582.

[75] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: vol. 25. 2012.

[76] Breanne P Christie et al. "Comparison of spike sorting and thresholding of voltage waveforms for intracortical brain–machine interface performance". In: *Journal of Neural Engineering* 12.1 (2015), p. 016009. ISSN: 1741-2552. DOI: 10.1088/1741-2560/12/1/016009.

[77] Felix Pei et al. "Neural Latents Benchmark '21: Evaluating latent variable models of neural population activity". In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. 2021. DOI: 10.48550/arxiv.2109.04463.

[78] Hansem Sohn et al. "Bayesian Computation through Cortical Latent Dynamics". In: *Neuron* 103.5 (2019), 934–947.e5. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2019.06.012.

[79] Trung Le and Eli Shlizerman. "STNDT: Modeling Neural Population Activity with a Spatiotemporal Transformer". In: *arXiv* (2022). DOI: 10.48550/arxiv.2206.04727. eprint: 2206.04727.

[80] Patrick T. Sadtler et al. "Neural constraints on learning". In: *Nature* 512.7515 (2014), pp. 423–426. ISSN: 0028-0836. DOI: 10.1038/nature13665.

[81] Emily R. Oby et al. "New neural activity patterns emerge with long-term learning". In: *Proceedings of the National Academy of Sciences* 116.30 (2019), pp. 15210–15215. ISSN: 0027-8424. DOI: 10.1073/pnas.1820296116.

[82] Gal Mishne et al. "Hierarchical Coupled-Geometry Analysis for Neuronal Structure and Activity Pattern Discovery". In: *IEEE Journal of Selected Topics in Signal Processing* 10.7 (2016), pp. 1238–1253. ISSN: 1932-4553. DOI: 10.1109/jstsp.2016.2602061. eprint: 1511.02086.

[83] Juan A. Gallego et al. "Neural Manifolds for the Control of Movement". In: *Neuron* 94.5 (2017), pp. 978–984. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2017.05.025.

[84] Rishidev Chaudhuri et al. "The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep". In: *Nature Neuroscience* 22.9 (2019), pp. 1512–1520. ISSN: 1097-6256. DOI: 10.1038/s41593-019-0460-x.

[85] Ege Altan et al. "Estimating the dimensionality of the manifold underlying multi-electrode neural recordings". In: *PLoS Computational Biology* 17.11 (2021), e1008591. ISSN: 1553-734X. DOI: 10.1371/journal.pcbi.1008591.

[86] Jeffrey S. Seely et al. "Tensor Analysis Reveals Distinct Population Structure that Parallels the Different Computational Roles of Areas M1 and V1". In: *PLoS Computational Biology* 12.11 (2016), e1005164. ISSN: 1553-734X. DOI: 10.1371/journal.pcbi.1005164.

[87] Peiran Gao et al. "A theory of multineuronal dimensionality, dynamics and measurement". In: *bioRxiv* (2017), p. 214262. DOI: 10.1101/214262.

[88] Guillaume Hennequin, Tim P. Vogels, and Wulfram Gerstner. "Optimal Control of Transient Dynamics in Balanced Networks Supports Generation of Complex Movements". In: *Neuron* 82.6 (2014), pp. 1394–1406. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2014.04.045.

[89] David Sussillo et al. "A neural network that finds a naturalistic solution for the production of muscle activity". In: *Nature Neuroscience* 18.7 (2015), pp. 1025–1033. ISSN: 1097-6256. DOI: 10.1038/nn.4042.

[90] Vishwa Goudar and Dean V Buonomano. "Encoding sensory and motor patterns as time-invariant trajectories in recurrent neural networks". In: *eLife* 7 (2018), e31134. DOI: 10.7554/elife.31134.

[91] Andrew Miri et al. "Behaviorally Selective Engagement of Short-Latency Effector Pathways by Motor Cortex". In: *Neuron* 95.3 (2017), 683–696.e11. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2017.06.042.

[92] Eric Trautmann et al. "Motor Cortex Isolates Skill-Specific Dynamics in a Context Switching Task". In: *Computational and Systems Neuroscience (COSYNE) Abstracts*. Lisbon, Portugal, 2022.

[93] Claire L. Warriner et al. "Motor cortical influence relies on task-specific activity covariation". In: *Cell Reports* 40.13 (2022), p. 111427. ISSN: 2211-1247. DOI: 10.1016/j.celrep.2022.111427.

[94] Marius Pachitariu, Shashwat Sridhar, and Carsen Stringer. "Solving the spike sorting problem with Kilosort". In: *bioRxiv* (2023), p. 2023.01.07.523036. DOI: 10.1101/2023.01.07.523036.

[95] Alex H. Williams et al. "Discovering Precise Temporal Patterns in Large-Scale Neural Recordings through Robust and Interpretable Time Warping". In: *Neuron* 105.2 (2020), 246–259.e8. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2019.10.020.

[96] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552.

[97] Frank Rosenblatt. *Principles of neurodynamics. Perceptrons and the theory of brain mechanisms*. Tech. rep. Buffalo, NY: Cornell Aeronautical Laboratory, 1961.

[98] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv* (2014). DOI: 10.48550/arxiv.1412.6980. eprint: 1412.6980.

[99] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *arXiv* (2014). DOI: 10.48550/arxiv.1406.1078. eprint: 1406.1078.

[100] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* (2012).

Methods

MINT (Part I): Estimating candidate states

Model of idealized trajectories

Let $\mathbf{s}_t \in \mathcal{S}^N$ where $\mathcal{S} = \{0, 1, \ldots, S\}$ be the measured spiking activity from N neurons at time sample t. Spikes are associated with some underlying neural state $\mathbf{x}_t \in \mathbb{R}^N$ and a corresponding behavioral state $\mathbf{z}_t \in \mathbb{R}^M$ where M is the number of behavioral variables (e.g. x- and y-velocity of the hand). We bin spiking activity in time such that $\bar{\mathbf{s}}_{i'} = \sum_{i=0}^{\Delta-1} \mathbf{s}_{\Delta i' - i}$. Given recent spiking history $\bar{\mathbf{s}}_{t'-\tau':t'}$ (spike counts from the most recently completed bin $t' = \lfloor t/\Delta \rfloor$ and τ' previous bins), we are interested in inferring posterior distributions over neural states $P(\mathbf{x}_t | \bar{\mathbf{s}}_{t'-\tau':t'})$ and over behavioral states $P(\mathbf{z}_t | \bar{\mathbf{s}}_{t'-\tau':t'})$. We can then perform *maximum a posteriori* estimation to generate candidate estimates for both the neural state and behavior. We introduce a model of the form:

$$\mathbf{x}_t \sim \mathrm{Unif}(\Omega) \tag{1}$$

$$\mathbf{x}_{t-i-1} = g(\mathbf{x}_{t-i}),\ i \in \{0, 1, \ldots, \tau - 1\} \tag{2}$$

$$\bar{\mathbf{s}}_{t'-i'} \sim \mathrm{Pois}\left(\frac{1}{\Delta} \sum_{i=0}^{\Delta-1} \mathbf{x}_{\Delta(t'-i')-i}\right),\ i' \in \{0, 1, \ldots, \tau'\} \tag{3}$$

$$\mathbf{z}_t = f(\mathbf{x}_t) \tag{4}$$

where $g : \Omega^+ \to \Omega^+$ is a state-transition lookup table, $f : \Omega \to \Phi$ is a lookup table that associates neural states with behavioral states, $\delta = t - \Delta t'$ is the number of samples elapsed since the last time bin completed, and $\tau = \delta + \Delta(\tau' + 1) - 1$ indicates the extent of the deterministic history for $\mathbf{x}_t$ (matching the extent of recent spiking history). This model has parameters Ω^+ and Φ. These parameters are described below, along with definitions for g and f.

Ω^+ is a set (library) of C neural trajectories, with each trajectory consisting of an ordered set of neural states. Each neural state in the library is notated as $\overline{\mathbf{x}}_k^c$, where c indexes trajectories and k indexes locations along each trajectory (e.g. $c = 2$ and $k = 100$ indicates the 100th neural state along the second trajectory in the library). Each trajectory is presumed to contain the sequence of neural states that are traversed for a particular behavior. Thus, k increasing along a neural trajectory corresponds to the passage of time during the execution of a behavior. Eq. (2) and Eq. (3) indicate that each $\mathbf{x}_t$ has a history of τ states that are responsible for generating recent spike count observations. However, the first τ states along each neural trajectory lack a complete state history. Thus, we learn the full Ω^+, but in Eq. (1) we only consider a reduced set of neural states, $\Omega = \left\{\overline{\mathbf{x}}_k^c \in \Omega^+ \mid k > \tau\right\}$, because only these states have the necessary state histories. The state-transition lookup table is defined as follows,

$$g(\overline{\mathbf{x}}_k^c) = \overline{\mathbf{x}}_{k-1}^c \tag{5}$$

simply indicating that each neural state in the library has a recent history determined by the stereotyped ordering of neural states within each neural trajectory.

Φ is a set (library) of C behavioral trajectories, where each trajectory consists of an ordered set of behavioral states. Each $\overline{\mathbf{x}}_k^c \in \Omega$ is associated with a behavioral state, $\overline{\mathbf{z}}_k^c$, via

$$f(\overline{\mathbf{x}}_k^c) = \overline{\mathbf{z}}_k^c \tag{6}$$

In other words, each neural state in the library is paired with a behavioral state for the same c and k. Once the libraries of neural and behavioral trajectories have been learned, MINT's parameters are fully learned. g will be determined by the ordering of states within each trajectory and f will be determined by each state's c and k indices. There are a variety of methods one can employ for learning these trajectories. The methods used in this paper are described in Learning idealized trajectories. These trajectories can often be learned by averaging neural and behavioral data across many repeated trials of the same movement, yielding $\overline{\mathbf{x}}_k^c$ and $\overline{\mathbf{z}}_k^c$ that are highly representative of the neural and behavioral states expected for a particular movement.

We assume $\overline{\mathbf{z}}_{k_1}^{c_1} = \overline{\mathbf{z}}_{k_2}^{c_2}$ if and only if $c_1 = c_2$ and $k_1 = k_2$. (We similarly assume $\overline{\mathbf{x}}_{k_1}^{c_1} = \overline{\mathbf{x}}_{k_2}^{c_2}$ if and only if $c_1 = c_2$ and $k_1 = k_2$.) This can be made trivially true by letting c and k be the first two behavioral variables. Thus, f is invertible. This trick is mathematically convenient for subsequent derivations—it does not change that multiple neural states can be associated with the same behavior as in Figure 1.

Estimating candidate states

The prior in Eq. (1) ensures that $P(\mathbf{x}_t = \mathbf{x} | \bar{\mathbf{s}}_{t'-\tau':t'}) = 0$ if $\mathbf{x} \notin \Omega$. Every element of Ω can be written as $\overline{\mathbf{x}}_k^c$ for some c and k, and the prior in Eq. (1) is uniform over the states in Ω. Eq. (2) and Eq. (5) indicate that each $\overline{\mathbf{x}}_k^c$ has only one possible recent history, a history that is specified by the stereotyped ordering of neural states in the c-th

trajectory. Thus, the posterior probabilities over neural states in Ω do not need to marginalize over multiple potential state histories. Rather, the probability associated with each $\overline{\mathbf{x}}_k^c$ is proportional to the likelihood of recently observed spikes, $\overline{\mathbf{s}}_{t'-i':t'}$, given the unique history of neural states associated with $\overline{\mathbf{x}}_k^c$. Thus, the posterior probabilities over neural states in Ω can be written

$$P(\mathbf{x}_t = \overline{\mathbf{x}}_k^c | \overline{\mathbf{s}}_{t'-\tau':t'}) \propto \prod_{i'=0}^{\tau'} \prod_{n=1}^{N} h(\overline{s}_{n,t'-i'}, \lambda_{n,k-\delta-\Delta i'}^c) \tag{7}$$

$$\lambda_{n,j}^c = \frac{1}{\Delta} \sum_{i=0}^{\Delta-1} \overline{x}_{n,j-i}^c \tag{8}$$

where $h(s,\lambda) = \frac{\lambda^s e^{-\lambda}}{s!}$ (i.e. Poisson likelihoods, as dictated by Eq. (3)), and $\overline{s}_{n,i'}$ and $\overline{x}_{n,i}^c$ are the n-th components of $\overline{\mathbf{s}}_{i'}$ and $\overline{\mathbf{x}}_i^c$, respectively. Recall that δ is the number of time samples elapsed since the completion of the most recent time bin. A normalizing constant can convert Eq. (7) into full posterior probabilities. Computing this constant is straightforward because Ω is finite.

Eq. (4) and Eq. (6) allow the posterior over behavioral states to be written in terms of the posterior over neural states. $P(\mathbf{z}_t = \mathbf{z} | \overline{\mathbf{s}}_{t'-\tau':t'}) = 0$ if $\mathbf{z} \notin \Phi$, but for behaviors in Φ the posterior probabilities are

$$P(\mathbf{z}_t = \overline{\mathbf{z}}_k^c | \overline{\mathbf{s}}_{t'-\tau':t'}) = P(\mathbf{x}_t = f^{-1}(\overline{\mathbf{z}}_k^c) | \overline{\mathbf{s}}_{t'-\tau':t'}) \tag{9}$$

We can perform *maximum a posteriori* estimation on the log-transformed neural state posterior and read out the behavioral estimate directly.

$$\hat{\mathbf{x}}_t = \underset{\overline{\mathbf{x}}_k^c \in \Omega}{\operatorname{argmax}} \sum_{i'=0}^{\tau'} \sum_{n=1}^{N} \log h(\overline{s}_{n,t'-i'}, \lambda_{n,k-\delta-\Delta i'}^c) \tag{10}$$

$$\hat{\mathbf{z}}_t = f(\hat{\mathbf{x}}_t) \tag{11}$$

Lookup table of log-likelihoods

One could proceed with querying Eq. (10) for all $\overline{\mathbf{x}}_k^c \in \Omega$, selecting the most probable neural state, then reading out the associated behavioral state with Eq. (11). However, one may often wish to speed up this process, e.g. to deploy in a real-time application. In this section (and the following two sections), we describe a procedure for approximating Eq. (10) with considerably reduced computational burden.

Notice that the log-likelihood term in Eq. (10) only varies as a function of spike count and rate. Spike counts are finite and firing rates have limited dynamic ranges, which can be discretized, so it is possible to precompute and store these log-likelihoods in a lookup table in memory. Suppose the dynamic range of rates is given by $[\lambda_{min}, \lambda_{max}]$ and we sample this range with $V+1$ values such that $\tilde{\lambda}(v) = \lambda_{min} + \frac{v}{V}(\lambda_{max} - \lambda_{min})$ for $v \in \mathcal{V}$ where $\mathcal{V} = \{0, 1, \ldots, V\}$. Every rate $\lambda_{n,j}^c$ can now be approximated by $\tilde{\lambda}(v_{n,j}^c)$ for some $v_{n,j}^c \in \mathcal{V}$ that minimizes the approximation error. If we define a lookup table with entries $L(s,v) = \log h(s, \tilde{\lambda}(v))$, then Eq. (10) can be rewritten

$$\hat{\mathbf{x}}_t = \underset{\overline{\mathbf{x}}_k^c \in \Omega}{\operatorname{argmax}} \sum_{i'=0}^{\tau'} \sum_{n=1}^{N} L(\overline{s}_{n,t'-i'}, v_{n,k-\delta-\Delta i'}^c) \tag{12}$$

Thus, during training we can compute L for all s and v. Similarly, we can compute $v_{n,j}^c$ for all c, n, and j. This formulation ensures the costly process of computing log-likelihoods only needs to be performed once, during training.

Recursive solution

The estimation procedure in Eq. (12) can be made faster still by exploiting recursive structure. Consider the following definitions.

$$q_{t,k}^c = \sum_{i'=0}^{\tau'} r_{t'-i',k-\delta-\Delta i'}^c \tag{13}$$

$$r_{j',j}^c = \begin{cases} \sum_{n=1}^{N} L(\overline{s}_{n,j'}, v_{n,j}^c) & j' > 0, j > 0 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

This expression can be generated recursively,

$$q_{t,k}^c = \begin{cases} q_{t-\Delta,k-\Delta}^c + r_{t',k}^c - r_{t'-\tau'-1,k-\tau-1}^c, & \delta = 0 \\ q_{t-1,k-1}^c, & \delta > 0 \end{cases} \tag{15}$$

with initial conditions $q^c_{t,0} = q^c_{0,k} = 0$. Given that $q^c_{t,k}$ is simply shifted by one index relative to the previous time step when $\delta > 0$, the state estimate can be written

$$\hat{\mathbf{x}}_t = \begin{cases} \underset{\overline{\mathbf{x}}^c_k \in \Omega}{\operatorname{argmax}}\, q^c_{t,k}, & \delta = 0 \\ \overline{\mathbf{x}}^{\hat{c}}_{\hat{k}+\delta}, & \delta > 0 \end{cases} \tag{16}$$

where $\hat{c}$ and $\hat{k}$ are the indices such that $\hat{\mathbf{x}}_{t-\delta} = \overline{\mathbf{x}}^{\hat{c}}_{\hat{k}}$. Note that $\hat{\mathbf{x}}_t$ cannot be computed until $t > \tau$ (when sufficient spiking history has been collected).

Querying fewer states

The approach described in Eq. (16) is efficient insofar as it renders a new estimate at each time t while only requiring substantial computations every Δ time samples. Nevertheless, when $\delta = 0$, the recursion requires computing new $r^c_{t',k}$ and $r^c_{t'-\tau'-1,k-\tau-1}$ for every element in Ω^+ to ensure that $q^c_{t,k}$ is defined for every $\overline{\mathbf{x}}^c_k \in \Omega$. However, we typically assume some smoothness in neural trajectories over time. Thus, for reasonably small Δ, we can approximate the most likely neural state when $\delta = 0$ by only considering a subset of neural states defined by

$$\tilde{\Omega} = \left\{ \overline{\mathbf{x}}^c_k \in \Omega \;\middle|\; \frac{k}{\Delta} \in \mathbb{Z} \right\} \tag{17}$$

which is equivalent to downsampling the neural trajectories in Ω with Δ spacing between samples. Letting $k' = \frac{k}{\Delta}$, this choice yields a new expression for the neural state estimate,

$$\hat{\mathbf{x}}_t = \begin{cases} \underset{\overline{\mathbf{x}}^c_k \in \tilde{\Omega}}{\operatorname{argmax}}\, \tilde{q}^c_{t',k'}, & \delta = 0 \\ \overline{\mathbf{x}}^{\hat{c}}_{\hat{k}+\delta}, & \delta > 0 \end{cases} \tag{18}$$

where $\tilde{q}^c_{t',k'} = q^c_{\Delta t',\Delta k'}$. The recursion can now be performed over fewer states.

$$\tilde{q}^c_{t',k'} = \tilde{q}^c_{t'-1,k'-1} + r^c_{t',\Delta k'} - r^c_{t'-\tau'-1,\Delta k'-\tau-1} \tag{19}$$

This simplification reduces memory and execution time. Furthermore, by using interpolation (described in the next section) we can still estimate neural states outside of $\tilde{\Omega}$.

Generating estimates at higher-than-bin-width resolution

Eq. (18) describes a procedure for generating new neural state estimates at every time sample. When $\delta = 0$, the estimate is updated using new spiking observations. When $\delta > 0$, the estimate is updated using the model of stereotyped trajectories, which specifies which neural state is likely to come next even in the absence of new spiking observations. However, to implement these equations in real time requires that the most time-consuming computations (those performed when $\delta = 0$) be performed within one time sample (e.g. 1 ms). MINT is very fast and in many cases will be capable of generating estimates in under a millisecond (Table 1). When this isn't possible, a small decoding lag can be introduced (e.g. 5 ms) such that the computations required when $\delta = 0$ can begin prior to needing the result.

MINT (Part II): Interpolation

The algorithm described above leverages a library of neural trajectories composed of discrete sequences of neural states typically observed for a discrete set of conditions. These trajectories are presumed to sample an underlying neural manifold that is fundamentally continuous. The library of neural trajectories will more finely sample that manifold if one uses a large C such that small variations on a behavior each get their own neural and behavioral trajectories. Additionally, recall from Querying fewer states that, while decoding, only a subset of neural states (spaced apart by Δ) are considered. The larger Δ is, the more likely it is that the neural state estimate will make a small error in time (estimating a neural state that is slightly ahead or slightly behind the true state along a trajectory). Thus, neural state estimation can be more accurate in time and reflect more behavioral variability if one uses a small Δ and a large C. However, the computational burden of the algorithm grows as Δ shrinks or C grows. (The training data requirement also increases with C). This creates a potential trade-off between performance and computational burden. However, MINT side-steps this trade-off by using linear interpolation between states.

When the neural manifold is presumed to smoothly vary between neural states (across time or conditions) and there is also smooth variation between the corresponding behavioral states, it is unnecessary to use an overly small Δ and/or large C. Instead, one can identify candidate states along a small set of neural trajectories that

coarsely sample the manifold, and then interpolate between those candidate states to find an intermediate state that is more consistent with the observed spikes. This two-step procedure (identify candidate states and then interpolate) allows the neural state estimate (and corresponding behavioral state estimate) to be accurate in time and capture variability between conditions (effectively as though a smaller Δ and larger C had been used) while preserving the computational benefits of a reasonable Δ (e.g. 20 ms) and small C.

Model of interpolated states

Suppose we identify two neural states, $\overline{\mathbf{x}}_{k_1}^{c_1}$ and $\overline{\mathbf{x}}_{k_2}^{c_2}$, and we'd like to consider the possibility that the actual neural state is somewhere between them. We can use a model of the following form:

$$\alpha \sim \mathrm{Unif}(0,1) \tag{20}$$

$$\mathbf{x}_{t-i} = (1-\alpha)\overline{\mathbf{x}}_{k_1-i}^{c_1} + \alpha\overline{\mathbf{x}}_{k_2-i}^{c_2},\ i \in \{0,1,\ldots,\tau\} \tag{21}$$

$$\overline{\mathbf{s}}_{t'-i'} \sim \mathrm{Pois}\left(\frac{1}{\Delta}\sum_{i=0}^{\Delta-1}\mathbf{x}_{\Delta(t'-i')-i}\right),\ i' \in \{0,1,\ldots,\tau'\} \tag{22}$$

$$\mathbf{z}_t = (1-\alpha)\overline{\mathbf{z}}_{k_1}^{c_1} + \alpha\overline{\mathbf{z}}_{k_2}^{c_2} \tag{23}$$

This model assumes that as the neural state smoothly varies from $\overline{\mathbf{x}}_{k_1}^{c_1}$ to $\overline{\mathbf{x}}_{k_2}^{c_2}$, the corresponding behavior smoothly varies from $\overline{\mathbf{z}}_{k_1}^{c_1}$ to $\overline{\mathbf{z}}_{k_2}^{c_2}$.

Interpolating between states

Explicitly evaluating the probabilities associated with all $\mathbf{x}_t$ in this model would be computationally inefficient. Given that the prior on α is uniform over the range $[0,1]$, we can simply select the α that maximizes the log-likelihood of the observed spikes. The log-likelihood of the observed spikes is given by the equation:

$$\breve{q}_t(\alpha) = \log P(\overline{\mathbf{s}}_{t'-\tau':t'}|\alpha) = \sum_{i'=0}^{\tau'}\sum_{n=1}^{N}\log h\left(\overline{s}_{n,t'-i'}, (1-\alpha)\lambda_{n,k_1-\delta-\Delta i'}^{c_1} + \alpha\lambda_{n,k_2-\delta-\Delta i'}^{c_2}\right) \tag{24}$$

Differentiating $\breve{q}_t$ twice with respect to α yields

$$\frac{d^2\breve{q}_t}{d\alpha^2} = -\sum_{i'=0}^{\tau'}\sum_{n=1}^{N}\overline{s}_{n,t'-i'}\left(\frac{\lambda_{n,k_2-\delta-\Delta i'}^{c_2} - \lambda_{n,k_1-\delta-\Delta i'}^{c_1}}{(1-\alpha)\lambda_{n,k_1-\delta-\Delta i'}^{c_1} + \alpha\lambda_{n,k_2-\delta-\Delta i'}^{c_2}}\right)^2 \tag{25}$$

Notice that $\frac{d^2\breve{q}_t}{d\alpha^2} \leq 0$ always, i.e. $\breve{q}_t$ is a concave function of α. Thus, we can rapidly compute $\hat{\alpha}$ using Newton's method and then estimate $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{z}}_t$ via Eq. (21) and Eq. (23). This procedure will yield the same $\hat{\alpha}$ for all t associated with the same t'. Thus, the interpolation only needs to be performed once per time bin, when $\delta = 0$. In this paper, we used the following stopping criteria for Newton's method: 1) the estimate of α is within $.01$ of the estimate at the previous iteration, 2) the estimate of α saturates at 0 or 1, or 3) optimization runs for 10 iterations.

Interpolating across indices

Eq. (17) restricts the neural states under consideration to be spaced apart by Δ indices. Thus, a straightforward use of the interpolation scheme described above is to interpolate between nearby neural states along the same trajectory to restore the precision in the estimate that was lost by this restriction. First, a candidate neural state, which we'll denote $\overline{\mathbf{x}}_{k_1}^{c_1}$, is identified using Eq. (18). Then, a second candidate neural state is identified, which we'll denote $\overline{\mathbf{x}}_{k_2}^{c_1}$. This second state is whichever of the two adjacent states (either $\overline{\mathbf{x}}_{k_1+\Delta}^{c_1}$ or $\overline{\mathbf{x}}_{k_1-\Delta}^{c_1}$) has a larger log-likelihood of the observed spikes. Then, interpolation between $\overline{\mathbf{x}}_{k_1}^{c_1}$ and $\overline{\mathbf{x}}_{k_2}^{c_1}$ is performed as in Interpolating between states to yield neural and behavioral state estimates $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{z}}_t$. This form of interpolation is less about generalization and more about enabling Eq. (17) to improve computational efficiency without loss of precision in the neural state estimates along each trajectory.

Interpolating across conditions

To interpolate across conditions, we also identify candidate neural states. The first candidate neural state, $\overline{\mathbf{x}}_{k_1}^{c_1}$, is again identified using Eq. (18). The second candidate neural state, $\overline{\mathbf{x}}_{k_2}^{c_2}$, is identified as the neural state that maximizes the log-likelihood of the observed spikes but is not a state along the same trajectory as $\overline{\mathbf{x}}_{k_1}^{c_1}$ (i.e. $c_2 \neq c_1$). Next, both $\overline{\mathbf{x}}_{k_1}^{c_1}$ and $\overline{\mathbf{x}}_{k_2}^{c_2}$ are each interpolated across indices with their adjacent states to yield improved candidate neural and behavioral state estimates. Then, an additional interpolation is performed between the improved candidate state estimates to yield the condition-interpolated neural and behavioral state estimates $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{z}}_t$. This allows for generalization between similar behaviors.

Other forms of interpolation

The exact manner in which interpolation is applied during MINT may vary by task or dataset. Often, interpolation across indices and conditions will be sufficient. However, in this section we'll review some alternative interpolation implementations that were used for various analyses throughout this paper.

First, a dataset like MC_RTT that lacks explicit condition structure will not necessarily have one trajectory per behavior. We used AutoLFADS to learn neural trajectories for MC_RTT, which would have yielded a single neural trajectory for the entire training set were it not for recording gaps that broke up that trajectory. Thus, different portions of the same trajectory frequently corresponded to similar movements that it made sense to consider interpolations between. We therefore modified interpolation such that the second candidate state could be on the same trajectory as the first candidate state, but it had to be at least 1000 milliseconds away from that first state. Additionally, in a task with many degrees of behavioral freedom, there may be many trajectory fragments near a candidate neural state that could be worthwhile to interpolate to. Thus, for the MC_RTT dataset we expanded to multiple candidate neural states (6 for decoding analyses, 5 for the neural state estimation analysis; these values were optimized as hyperparameters). Interpolation occurred between all pairs of candidate neural states. Then, we proceeded to only use the interpolation that yielded the highest spike count log-likelihoods. These modifications ensured that utilizing long trajectories spanning multiple behaviors in a reaching task with many degrees of behavioral freedom would not limit the ability of interpolation to generalize between similar behaviors.

Second, in DMFC_RSG we computed multiple libraries of neural trajectories, each library corresponding to a different section of the training data in which the overall firing rate statistics differed due to recording instabilities throughout the session. For this dataset, we wished to expand interpolation to occur across indices, conditions, *and* libraries. We simply extended the approach for interpolating across conditions one step further. The first candidate neural state was selected as the most likely neural state across all libraries. The second candidate neural state was selected as the most likely neural state in any library except the one in which the first candidate state resided. For each of these two candidate states, within-library interpolation across indices and conditions proceeded to improve the estimates. Then, a final interpolation occurred between the best estimate in the first library with the best estimate in the second library. This approach enabled MINT to robustly estimate the neural state even when fed test data from various sections of the dataset in which different recording instabilities were present.

Third, in MC_Cycle we decoded a circular variable: phase. To accommodate this circularity, phase interpolations were modified such that they always occurred across the acute angle between the two phases. For example, if two candidate states corresponding to phases of 10 degrees and 350 degrees were identified, interpolation occurred across the 20 degree difference rather than the 340 degree difference.

Datasets

We analyzed 8 empirical datasets (Area2_Bump, DMFC_RSG, MC_Cycle, MC_Maze, MC_Maze-L, MC_Maze-M, MC_Maze-S, MC_RTT), several simulated maze datasets, and an artificial multitask spiking network. All empirical datasets contained spike times from offline or online sorted units. Detailed descriptions of all empirical datasets (except MC_Cycle) can be found in Pei et al. [77]. Here, we briefly describe each of the datasets.

Dataset	Neurons	Conditions	Trials
Area2_Bump	65	16	364
DMFC_RSG	54	40	1006
MC_Cycle	112	8	273
MC_Maze	182	108	2295
MC_Maze-L	162	27	500
MC_Maze-M	152	27	250
MC_Maze-S	142	27	100
MC_RTT	130	N/A	1080
Maze Simulations	182	108	2295
Multitask Network	1200	9	270

Table 2. Neuron, condition, and trial counts for each dataset. For some datasets, there are additional trials that are excluded from these counts. These trials are excluded because they were only usable for Neural Latents Benchmark submissions due to hidden behavioral data and partially hidden spiking data (see Neural Latents Benchmark).

Area2_Bump

This dataset contains neural recordings (96-electrode Utah array) from Brodmann's area 2 of S1 (a proprioceptive area) while a monkey used a manipulandum to make center-out-reaches to one of eight targets. On some trials, the

monkey volitionally reached to the targets ('active' trials). On other trials, the manipulandum perturbed the monkey's arm out toward one of the targets ('passive' trials). Thus, right after movement onset the 'active' trials involved predictable sensory feedback (the monkey planned the movement) and the 'passive' trials involved unexpected sensory feedback (the monkey was not planning to move). Between the eight targets and two trial types ('active' and 'passive'), there were a total of 16 conditions. The behavioral data for this dataset includes: hand position (x- and y-components), hand velocity (x- and y-components), forces and torques applied to the manipulandum (x- y- and z-forces; x- y- and z-torques), 7 joint angles, 7 joint velocities, 39 muscle lengths, and 39 muscle velocities. Position data was zeroed at movement onset for each trial to ensure position data was relative to a pre-movement baseline. More information on the task and data can be found in Chowdhury, Glaser, and Miller [73].

DMFC_RSG

This dataset contains neural recordings (three Plexon probes) from dorsomedial frontal cortex while a monkey performed a time-interval reproduction task (aka Ready-Set-Go). In this task, the monkey received two visual cues, 'Ready' and 'Set', separated in time by a sample interval. The monkey was required to estimate the sample interval and report this estimate by performing an action ('Go') that followed the 'Set' cue by a production interval. The monkey was rewarded depending on how close the production interval was to the sample interval on a given trial. The sampling interval was selected on each trial from one of two prior distributions. The short prior distribution contained sampling intervals of 480, 560, 640, 720, and 800 ms. The long prior distribution contained sampling intervals of 800, 900, 1000, 1100, and 1200 ms. Trials were selected from prior distributions in blocks (i.e. many consecutive trials were drawn from the same prior distribution). Depending on the trial, the action could be reported in one of four ways: a joystick movement or an eye saccade to either the left or right. Between the two prior distributions, five sampling intervals per prior distribution, and four actions, there were a total of 40 conditions. More information on the task can be found in Sohn et al. [78].

MC_Cycle

This dataset contains neural recordings (two 96-electrode Utah arrays) from motor cortex (M1 and PMd) while a monkey performed a cycling task to navigate a virtual environment. Offline sorting of spike waveforms (using Kilosort [94]) yielded single-neuron and high-quality multi-neuron isolations. The dataset also includes threshold crossings that were detected online. On each trial, the monkey grasped a hand-pedal and moved it cyclically forward or backward to navigate the virtual environment. The color of the virtual landscape cued the monkey as to whether forward or backward pedaling would advance the avatar in the virtual environment ('green' landscape: forward pedaling advanced the avatar; 'tan' landscape: backward pedaling advanced the avatar). Pedaling distance was cued by a virtual target that appeared a fixed distance away in the virtual environment. The number of complete cycles of pedaling required to reach the target was either 1, 2, 4, or 7 cycles. Between the two pedaling directions and four pedaling distances, there were a total of 8 conditions. The behavioral data for this dataset includes: pedal phase, angular velocity of the pedal, x- and y- pedal position, x- and y- pedal velocity, and 7 intramuscular EMG recordings. The dataset includes both sorted spikes and threshold crossings. More information on the task can be found in Schroeder et al. [20].

MC_Maze

This dataset contains neural recordings (two 96-electrode Utah arrays) from motor cortex (M1 and PMd) while a monkey performed straight and curved reaches. The monkey was presented with a maze on a vertical screen with a cursor that tracked the motion of the monkey's hand. Targets appeared on the screen and the monkey had to make reaches that would move the cursor onto the target. Virtual barriers were presented that the cursor could not pass through, requiring curved reaches that avoided the barriers. Each maze configuration had three variants. The first variant had a single target with no barriers (straight reach). The second variant had a single target with a barrier (curved reach). The third variant had a target with a barrier (curved reach) as well as two unreachable distractor targets elsewhere in the maze. There were 36 maze configurations for a total of 108 conditions. The behavioral data for this dataset includes x- and y-components of hand position and velocity. Position data was zeroed at movement onset for each trial to ensure position data was relative to a pre-movement baseline. More information on the task can be found in Churchland et al. [72].

MC_Maze-(L,M,S)

These three datasets were collected from the same monkey as in MC_Maze performing the same task, but on different days with different conditions. The same arrays were used to collect neural recordings (though different numbers of sorted units were identified in each session) and the same behavioral data was collected. These datasets differ from MC_Maze primarily in that they have fewer conditions (nine maze configurations with three variants, for a total of 27 conditions) and fewer trials.

MC_RTT

This dataset contains neural recordings (96-electrode Utah array) from motor cortex while a monkey made reaches in a horizontal plane to targets in an 8 x 8 grid. The task had no explicit trial structure nor did it require that individual movements be repeated throughout the session. Rather, a target would appear in one of the 64 grid locations and the monkey would reach to that target. Then, a new target would appear at a different random grid location and the monkey would reach there. This process continued throughout the session, leading to a collection of reaches that varied in terms of reach direction, reach distance, and reach speed. Despite lacking meaningful trial structure, the session was nevertheless broken up into 600 ms 'trial' segments (hence the trial count listed in Table 2). The behavioral data for this dataset includes x- and y-components of finger position and velocity. Due to the structure of the training data, neural trajectories were learned in long stretches spanning multiple movements. Thus, the zeroing of position data at movement onset that was employed for other datasets would have led to discontinuities in the behavioral trajectories for this task. We could have decoded position with no zeroing – however, similar movements (with similar corresponding neural activity) could have very different starting and ending positions in this dataset. Thus, we decided to focus decoding analyses on velocity only. More information on the task can be found in Makin et al. [58].

Maze task simulations

Several synthetic datasets were generated based on the maze task. These simulations were based on the MC_Maze dataset and matched the neuron, condition, and trial counts from MC_Maze. In all cases, the actual behavior from MC_Maze was used — spiking activity was the only component that was simulated.

To simulate firing rates, we first z-scored hand position, velocity, and acceleration (x- and y-components). Then, we simulated rates for each neuron as a random weighted sum of these z-scored kinematics with a constant offset. The weights and offsets were scaled such that the means and standard deviations of firing rates for the simulated neurons matched the means and standard deviations of the empirical trial-averaged rates in MC_Maze. In some simulations, we increased simulated firing rates by a factor of 5 or 10.

Simulated spiking activity was generated from simulated rates in one of two ways. In one simulation, spiking activity was simulated as a Poisson process, which corresponds to exponential-interval spiking. In other simulations, spiking activity was simulated with gamma-interval spiking. More specifically, we let the interval between spikes be drawn from a gamma distribution with a shape parameter $\alpha = 2$ and a rate parameter $\beta = 2\lambda$, where λ is the simulated firing rate. For reference, exponential-interval spiking corresponds to a gamma distribution with $\alpha = 1$ and $\beta = \lambda$. The overall rate of spiking was matched regardless of whether exponential-interval or gamma-interval spiking was simulated, but spiking occurred more regularly given the same rate for the gamma-interval simulations.

Multitask spiking network

A simulated multitask dataset was generated using the method described in DePasquale et al. [33]. A spiking network was trained to generate posterior deltoid activity during reaching (eight reach conditions) and cycling (one cycling condition). In the network, there were 39 latent factors related to reaching and 12 latent factors related to cycling. The factors, and therefore the neural trajectories, for reaching and cycling were roughly orthogonal to one another. The network simulated 500 consecutive trials. A 135-trial stretch of this simulated dataset was designated as the test set ($\sim$7.1 minutes of data). The training set was constructed by randomly selecting 15 trials per condition (135 training trials total) from the remaining trials.

Learning idealized trajectories

When training data includes repeated trials of the same behavior, neural and behavioral trajectories can be learned via standard trial-averaging. This procedure begins with filtering spikes temporally (e.g. with a Gaussian a filter) to yield single-trial rates. Then, rates and behavioral variables are extracted on each trial relative to behaviorally relevant trial events (e.g. movement onset). Depending on the structure of the task, time may need to be warped to ensure that task epochs unfold on the same timescale across different trials of the same condition. Finally, the single-trial rates are averaged across trials of the same condition to yield firing rates. The vector of firing rates at each time within each condition defines a neural state and the collection of neural states within each condition defines a neural trajectory. Single-trial behavioral variables are similarly averaged across trials to yield behavioral states and trajectories.

In the sections below, we describe how this standard trial-averaging procedure was applied to the datasets in this paper. We describe an additional trial-smoothing step that can be applied prior to averaging that often yields a small performance boost for MINT. We also describe a procedure for smoothing across conditions and neurons after averaging that can improve performance. All trajectories were learned at millisecond resolution.

There are two datasets for which non-standard trajectory-learning procedures were utilized. First, the training set for MC_RTT does not contain repeated trials of the same behavior and therefore was not amenable

to the trial-averaging procedure. Trajectories were instead learned using AutoLFADS, as described in its own section below. Second, DMFC_RSG contained substantial recording instabilities. Thus, although this dataset still used trial-averaging, an additional procedure was developed for learning multiple libraries of neural trajectories corresponding to the same behaviors at different portions of the session in which different recording instabilities were present. This procedure is also described in its own section below.

Filtering, extracting, and warping data on each trial

First, spiking activity for each neuron on each trial was temporally filtered with a Gaussian to yield single-trial rates. Table 3 reports the Gaussian standard deviations σ (in milliseconds) used for each dataset. In general, datasets with fewer trials and/or noiser neural recordings tended to benefit from larger values.

Next, single-trial rates and behavioral data were extracted on each trial relative to key trial events. For example, in the maze datasets, data was extracted beginning 500 ms prior to movement onset and ending 700 ms after movement onset. These extraction boundaries (listed in Table 3) determine when the neural trajectories begin and end. These boundaries should be set to ensure that the behaviors one is interested in decoding are represented in the library of trajectories. It is also important to have extra data at the beginning of each trajectory because the first τ indices on each trajectory cannot be selected as candidate states (due to lack of sufficient state history).

In the maze datasets and in Area2_Bump, there was no need to warp time – all movements of the same condition unfolded over similar timescales relative to movement onset. However, in the cycling dataset this was not the case. The duration between movement onset and offset was variable across trials of the same condition. Thus, data in MC_Cycle was time-warped (separately for each condition) using the procedure described in Russo et al. [63]. In the DMFC_RSG dataset, the duration between the 'set' cue and 'go' action was variable across trials. Thus, uniform time-warping to match the average duration for each condition was used. More generally, one may find the time-warping technique in Williams et al. [95] to be a useful tool. It is important that warping take place *after* filtering spikes, because warping raw spiking activity would change the neurons' rates.

The result of filtering, extracting, and warping is that, for each condition c, single-trial rates can be formatted into a tensor $\mathbf{X}^c \in \mathbb{R}^{N \times K_c \times R}$ where N is the number of neurons, K_c is the number of extracted time points on each trial, and R is the number of trials. Similarly, behavioral data can be formatted into tensors $\mathbf{Z}^c \in \mathbb{R}^{M \times K_c \times R}$ where M is the number of behavioral variables.

Averaging across trials

It is straightforward to average each $\mathbf{X}^c$ across trials yielding matrices $\overline{\mathbf{X}}^c \in \mathbb{R}^{N \times K_c}$. We refer to this as Type I averaging. However, we found that a modified trial-averaging procedure (Type II) led to slightly improved neural state estimation and decoding in most cases. First, all rates ($\mathbf{X}^c$ for all c) are mean-centered and soft-normalized as described in Russo et al. [63]. The offset and scaling factor for this step are computed based on Type I averaged rates. Then, the mean-centered and soft-normalized rates are formatted into matrices $\mathbf{X}^c_{in} \in \mathbb{R}^{R \times NK_c}$. PCA is used to reduce the trial-dimensionality of the rates to 1 for each c via

$$\mathbf{X}^c_{out} = W_c W_c^\top \mathbf{X}^c_{in} \tag{26}$$

where $W_c \in \mathbb{R}^R$ is a column-vector corresponding to the first principal component of $\mathbf{X}^c_{in}$. Then, we reverse the soft-normalization and mean-centering operations on $\mathbf{X}^c_{out}$ and average across trials to get $\overline{\mathbf{X}}^c$ (Type II). Table 3 indicates for each dataset which trial-averaging procedure was used.

We average each $\mathbf{Z}^c$ across trials yielding matrices $\overline{\mathbf{Z}}^c \in \mathbb{R}^{M \times K_c}$. Typically, no modified trial-averaging procedure is required for behavioral data. However, there are cases where a behavioral variable should not be averaged without some pre- and/or post-processing. For example, in MC_Cycle, phase is a circular variable and therefore can't be linearly averaged. To accommodate this, we unwrapped phase, averaged across trials, then re-wrapped it.

Smoothing across neurons and/or conditions

To additionally improve trial-averaged rate estimates we can use dimensionality reduction to smooth across neurons and/or conditions. First, trial-averaged rates are mean-centered and soft-normalized. Then, they are concatenated across conditions into a matrix $\overline{\mathbf{X}}_{in} \in \mathbb{R}^{N \times K}$ where $K = \sum_c K_c$. PCA is used to reduce the neural-dimensionality of the rates via

$$\overline{\mathbf{X}}_{out} = WW^\top \overline{\mathbf{X}}_{in} \tag{27}$$

where $W \in \mathbb{R}^{N \times D_{neural}}$ is the top D_{neural} principal components of $\overline{\mathbf{X}}_{in}$. Then, $\overline{\mathbf{X}}_{out}$ is reformatted into a new $\overline{\mathbf{X}}_{in} \in \mathbb{R}^{C \times NK_c}$ where K_c is the same for all c (this form of smoothing cannot be applied when this is not the case). PCA is used to reduce the condition-dimensionality of the rates via

$$\overline{\mathbf{X}}_{out} = WW^\top \overline{\mathbf{X}}_{in} \tag{28}$$

where $W \in \mathbb{R}^{C \times D_{condition}}$ is the top $D_{condition}$ principal components of $\overline{\mathbf{X}}_{in}$. Then, we reverse the soft-normalization and mean-centering operations on $\overline{\mathbf{X}}_{out}$, rectify to ensure non-negative rates, and reformat the smoothed rates back into matrices $\overline{\tilde{\mathbf{X}}}^c \in \mathbb{R}^{N \times K_c}$. Not all datasets benefited from neuron smoothing and/or condition smoothing. The smoothing dimensionalities used for each dataset are reported in Table 3.

After completing the previous steps (some combination of smoothing rates over time, trials, neurons, and/or conditions), there will be matrices of firing rates $\overline{\tilde{\mathbf{X}}}^c$ and behavioral data $\overline{\mathbf{Z}}^c$ for each c. The neural and behavioral states comprising the idealized neural and behavioral trajectories are then directly defined by these matrices.

$$\overline{\mathbf{x}}_k^c = \overline{\tilde{\mathbf{X}}}^c_{:,k} \tag{29}$$

$$\overline{\mathbf{z}}_k^c = \overline{\mathbf{Z}}^c_{:,k} \tag{30}$$

Dataset	Trajectory start	Trajectory end	σ	Trial Averaging	D_{neural}	$D_{condition}$
Area2_Bump	t_{move} – 350	t_{move} + 750	25	Type II	Full	Full
DMFC_RSG	t_{go} – 1950	t_{go} + 750	55	Type I	49	17
MC_Cycle	t_{move} – 600	t_{stop} + 600	30	Type I	Full	N/A
MC_Maze	t_{move} – 500	t_{move} + 700	30	Type II	Full	21
MC_Maze-L	t_{move} – 500	t_{move} + 700	35	Type II	Full	16
MC_Maze-M	t_{move} – 500	t_{move} + 700	60	Type II	Full	20
MC_Maze-S	t_{move} – 500	t_{move} + 700	85	Type II	120	10
Maze Simulations	t_{move} – 500	t_{move} + 700	30	Type II	Full	21
Multitask Network	(C) t_{move} – 1000 (R) t_{move} – 800	(C) t_{stop} + 1000 (R) t_{move} + 1000	15	Type I	Full	N/A

Table 3. Hyperparameters for learning neural trajectories via standard trial-averaging. σ is the standard deviation of the Gaussian kernel used to temporally filter spikes. Trial-averaging Type I and Type II procedures are described in Averaging across trials. D_{neural} and $D_{condition}$ are the neural- and condition-dimensionalities described in Smoothing across neurons and/or conditions. 'Full' means that no dimensionality reduction was performed. Condition smoothing could not be performed for MC_Cycle or the multitask network because different conditions in these datasets are of different lengths (i.e. K_c is not the same for all c). (C) and (R) refer to the cycling and reaching trajectories, respectively, in the multitask network. t_{move}, t_{stop}, and t_{go} correspond to movement onset, movement offset, and the 'go' time in the ready-set-go task, respectively.

Learning trajectories for MC_RTT

Neural trajectories were learned for MC_RTT using AutoLFADS [39]. Following the procedure described in Keshtkaran et al. [39], spiking activity in the training data was formatted into 600 ms segments with spike counts binned every 5 ms. Each segment overlapped with the previous segment by 200 ms and with the subsequent segment by 200 ms. AutoLFADS was then run twice on this formatted data (using 80/20 training and validation splits), yielding two sets of rate estimates that we then averaged across. Each AutoLFADS run yields slightly different results. Thus, we reasoned that averaging across runs would improve performance by reducing the impact of idiosyncratic rate variability that doesn't show up consistently across runs. We chose not to average more than two runs due to the computational burden this would accrue for training.

After averaging across runs, the rates from each segment were stitched back together (via the weighted average described in Keshtkaran et al. [39]) into long neural trajectories spanning many movements. The trajectories were then upsampled to 1 kHz via linear interpolation. If there had been no recording gaps, this procedure would have yielded a single neural trajectory. For the decoding analyses, there were 2 recording gaps and therefore 3 neural trajectories. For the neural state estimation analysis (which utilized a larger training set), there were 3 recording gaps and therefore 4 neural trajectories. In all cases, each trajectory contained ∼2.7 minutes of data.

When running AutoLFADS for the neural state estimation analysis, one of the runs returned a highly oscillatory set of rates (∼25-40 Hz oscillations) that did not by eye resemble the output of the other AutoLFADS runs. Although we confirmed this set of rates performed similarly for neural state estimation (.200 bits/spike using the oscillatory rates alone, compared to .201 bits/spike when averaging across two non-oscillatory runs), we nevertheless chose to discard this run and average across two non-oscillatory AutoLFADS runs because the oscillatory solution was not one that AutoLFADS consistently returns for this particular training set.

See Other forms of interpolation for details on how these long neural trajectories are used during interpolation to improve neural state estimation and decoding.

Learning trajectories for DMFC_RSG

Learning neural trajectories for the DMFC_RSG dataset involved two main challenges. First, we needed to accommodate that each trial contained multiple trial events to align to. Second, we desired a procedure for creating multiple libraries of neural trajectories corresponding to different portions of the session in which different recording instabilities were present. The solutions we developed for each of these challenges are described below.

Each trial contained five main trial events: the 'fixation' time when the monkey began looking at the center of the screen, the 'target onset' time when a white target appeared in one of two locations on the screen, the 'ready' cue, the 'set' cue, and the 'go' time at which the monkey initiated a joystick movement or saccade (depending on the condition). Thus, when learning neural trajectories, we needed to compute rates for five trial epochs: fixation to target onset, target onset to ready cue, ready cue to set cue, set cue to go time, and go time to the end of the trial. The first three epochs could not be averaged across trials in the standard way (because the durations varied from one trial to the next), but it was also not appropriate to warp them. Warping should be performed when the monkey is executing a computation or behavior at a variable speed. In this case, the monkey was simply waiting for the next trial event without knowledge of when it would arrive. Thus, we chose to align to the initial event in each epoch, average across trials (ignoring missing data associated with the variable epoch durations), and then trim the averaged rates to the median epoch duration. In this task, certain conditions are not disambiguated for the monkey until the set cue arrives (e.g. the monkey doesn't know if they are in an 800 ms or 900 ms interval trial until the set cue arrives). Thus, averages for a given condition in these first three epochs included all trials from the given condition plus all trials from other conditions that could not be distinguished from the given condition at this stage of the trial. In the set cue to go time epoch, rates were uniformly warped to match the average duration within each condition and then averaged across trials. All data after the go time were aligned to the go time and averaged. After computing these epoch-specific rate averages, the rates were concatenated across epochs to yield neural trajectories for each condition. They were then trimmed according to the trajectory start and end times listed in Table 3.

With access to a very large dataset, computing multiple libraries of neural trajectories for different recording conditions is straightforward: simply break the training data up in time into different sections and separately compute a library of trajectories for each. However, given limited training data, we sought to implement a similar strategy while not limiting the trajectories in each section to be based only on the small amount of spiking data available in that section. Thus, we first learned a library of neural trajectories based on the entire session (following the procedure described above) and smoothed across neurons and conditions to improve this estimate (yielding $\overline{\tilde{\mathbf{X}}}^{c}$ for each condition c). Then, we broke up the session into 6 consecutive sections and proceeded to learn a complete set of neural trajectories for each section with no smoothing across neurons or conditions (yielding $\overline{\mathbf{Y}}^{c,d}$ for each condition c and section d). Finally, we learned a linear transformation of $\overline{\tilde{\mathbf{X}}}^{c}$ for each section d that would better match the transformed firing rates for each section to those in $\overline{\mathbf{Y}}^{c,d}$ while preserving some of the neural geometry learned at the level of the session. This occurred via the equation

$$\overline{\tilde{\mathbf{Y}}}^{c,d} = (W^{(d)}\Lambda + I)\overline{\tilde{\mathbf{X}}}^{c} + b^{(d)} \tag{31}$$

where $W^{(d)} \in \mathbb{R}^{N\times N}$, $b^{(d)} \in \mathbb{R}^{N}$, and $\Lambda \in \mathbb{R}^{N\times N}$. Here, Λ is simply a diagonal matrix whose action on $\overline{\tilde{\mathbf{X}}}^{c}$ is to soft-normalize. The values in Λ are learned from $\overline{\tilde{\mathbf{X}}}^{c}$ via the same soft-normalization procedure described in Russo et al. [63] and used previously in this paper. The linear transformation was formulated in Eq. (31) to ensure that all regularization when learning the weights only applied to the deviation between the session-wide rates and the section-specific transformed rates. $W^{(d)}$ and $b^{(d)}$ were learned via an L2-regularized weighted least squares regression,

$$W^{(d)} = Q^{(d)}SP^{\top}(PSP^{\top} + \lambda I)^{-1} \tag{32}$$

$$b^{(d)} = \boldsymbol{\mu}_1^{(d)} - W^{(d)}\boldsymbol{\mu}_2 \tag{33}$$

$$Q^{(d)} = \overline{\mathbf{Y}}^{d} - \overline{\tilde{\mathbf{X}}} - \boldsymbol{\mu}_1^{(d)} \tag{34}$$

$$P = \Lambda\overline{\tilde{\mathbf{X}}} - \boldsymbol{\mu}_2 \tag{35}$$

$$\overline{\mathbf{Y}}^{d} = \begin{bmatrix} \overline{\mathbf{Y}}^{1,d} & \overline{\mathbf{Y}}^{2,d} & \dots & \overline{\mathbf{Y}}^{C,d} \end{bmatrix} \tag{36}$$

$$\overline{\tilde{\mathbf{X}}} = \begin{bmatrix} \overline{\tilde{\mathbf{X}}}^{1} & \overline{\tilde{\mathbf{X}}}^{2} & \dots & \overline{\tilde{\mathbf{X}}}^{C} \end{bmatrix} \tag{37}$$

where S is a diagonal matrix of observation weights, $\boldsymbol{\mu}_1^{(d)}$ is the mean of $\overline{\mathbf{Y}}^{d} - \overline{\tilde{\mathbf{X}}}$ across columns, and $\boldsymbol{\mu}_2$ is the mean of $\Lambda\overline{\tilde{\mathbf{X}}}$ across columns. In Eq. (34) and Eq. (35), the subtractions of $\boldsymbol{\mu}_1^{(d)}$ and $\boldsymbol{\mu}_2$ are applied to each column of the matrices they subtract from. The diagonal entries of S weight how much each observation should

matter when learning $W^{(d)}$. Given that the epoch between the set cue and the go time is when the monkey is performing an internal computation (keeping track of elapsed time internally) and this epoch is a large fraction of the evaluated trial period in the Neural Latents Benchmark, we decided $W^{(b)}$ should prioritize fitting this epoch well. Thus, diagonal entries of S corresponding to samples in the set-go epoch were given a weight s_{set-go} whereas all other diagonal entries of S were set to 1. Both the set-go epoch weight and the L2 regularization term were treated as hyperparameters and optimized on a validation set, yielding $s_{set-go} = 4$ and $\lambda = 100$. To summarize, $W^{(d)}$ transforms the mean-centered, soft-normalized session-wide rates into a set of rate residuals that can be added to the session-wide rates to better match the firing rate statistics in each section of the data. The result of this procedure is a complete library of neural trajectories for each section of the training data d.

$$\overline{\mathbf{x}}_k^{c,d} = \overline{\tilde{\mathbf{Y}}}_{:,k}^{c,d} \tag{38}$$

See Other forms of interpolation for details on how these multiple libraries are used during interpolation to improve the neural state estimate. There are no behavioral trajectories for this dataset (it was only used for neural state estimation analyses).

Visualizing neural trajectories

Neural trajectories have dimensionality matching the number of recorded neurons (or multi-units), but it is often useful to visualize them in a 2D state space. Throughout this paper, PCA is used to project neural trajectories into a low-dimensional subspace. When this done, firing rates are preprocessed with mean-centering and soft-normalization (as in Russo et al. [63]). In some cases, it is useful to rotate the data within the top PCs to find a perspective that highlights certain properties of the trajectories. When this is done, we report the neural variance captured by the plotted dimensions (along with the neural variance captured by the top PCs) to contextualize the scale of the neural dimensions. Percent neural variance captured was computed in the standard way, as described in Schroeder et al. [20].

Distance analyses

In Figure 1c-e, several distance metrics were used to analyze the MC_Cycle dataset. Neural distance is defined as the Euclidean distance between two neural states in the full-dimensional space. Muscle distance is defined by z-scoring the 'muscle state' (portion of behavioral state corresponding to muscle activity), then computing the Euclidean distance between two normalized muscle states. The muscle state consisted of seven intramuscular EMG recordings (long head of the biceps, anterior deltoid, lateral deltoid, posterior deltoid, trapezius, lateral tricep, and long head of the triceps). Kinematic distance is defined using the phases and angular velocities associated with two behavioral states. Both phase and angular velocity are z-scored (for phase, this utilizes the circular standard deviation). Then, the phase distance (d_{ph}) is computed as the circular distance between the two normalized phases. The angular velocity distance (d_{av}) is computed as the absolute difference between the two normalized angular velocities. Then, the kinematic distance is computed as $d_{kin} = \sqrt{d_{ph}^2 + d_{av}^2}$. All trajectories (neural and behavioral) were binned at 10 ms resolution prior to computing pairwise distances to keep the analysis computationally manageable. In Figure 1e, data was partitioned into two trial sets ('Partition A' and 'Partition B') for a control analysis. To keep the number of trials used to compute trajectories matched across all analyses in Figure 1c-e, only trials from 'Partition A' were utilized in the neural distance vs. muscle distance and neural distance vs. kinematic distance analyses. When plotting pairwise distances, all distances were normalized (separately for each plotting axis) such that 1 corresponded to the average pairwise distance.

Decoding analyses

All decoding analyses utilized the train-test trial splits listed in Table 4. Neural and behavioral trajectories were learned from the training set as described in Learning idealized trajectories. Then, MINT was provided with spiking activity on test trials with which to decode behavior. The trial epochs over which performance was evaluated were set to match the evaluation epochs from the Neural Latents Benchmark (for datasets that were used in the benchmark) and are listed in Table 4.

MINT used a bin size of Δ = 20 ms. The amount of spiking history provided to MINT at each decoding time step varied by dataset and is listed in Table 4. Decoding was always causal (i.e. only utilizing spiking history prior to the decoded moment), with one exception described below in MINT variants. MINT utilized interpolation for all datasets. The lookup table of log-likelihoods utilized a minimum rate, λ_{min}, corresponding to 1 spike/second. Log-likelihoods in the table were also clipped so as not to drop below $\log(10^{-6})$. These two choices regularize decoding such that a spurious spike from a neuron whose rate is close to 0 cannot dominate the decision of which neural state is most probable.

Dataset	Training trials	Test trials	Evaluation start	Evaluation end	Window length (ms)
Area2_Bump	272	92	t_{move} – 100	t_{move} + 500	240
MC_Cycle	174	99	t_{move} – 250	t_{stop} + 250	200
MC_Maze	1721	574	t_{move} – 250	t_{move} + 450	300
MC_Maze-L	375	125	t_{move} – 250	t_{move} + 450	300
MC_Maze-M	188	62	t_{move} – 250	t_{move} + 450	300
MC_Maze-S	75	25	t_{move} – 250	t_{move} + 450	340
MC_RTT	810	268	t_{start}	t_{start} + 599	480
Maze Simulations	1721	574	t_{move} – 250	t_{move} + 450	300
Multitask Network	135	135	t_{start} (first trial)	t_{end} (last trial)	300

Table 4. Details for decoding analyses. The number of training and testing trials for each dataset are provided along with the evaluation period over which performance was computed. The window lengths refer to the amount of spiking history MINT used for decoding (e.g. when $\tau' = 14$ and $\Delta = 20$, the window length is $(\tau' + 1)\Delta = 300$ ms). t_{move} corresponds to movement onset, t_{stop} corresponds to movement offset, t_{start} refers to the beginning of a trial, and t_{end} refers to the end of a trial. There is no defined condition structure for MC_RTT to use for defining trial boundaries. Thus, each trial is simply a 600 ms segment of data, with no alignment to movement. Although 270 of these segments were available for testing, the first 2 segments lacked sufficient spiking history for all decoders to be evaluated and were therefore excluded, leaving 268 test trials. For the multitask network, performance was evaluated on a continuous stretch of 135 trials spanning ∼7.1 minutes.

Decoding R^2

Decoded behavioral variables were compared to ground truth behavioral variables over the evaluation epoch of all test trials. Performance was reported as the coefficient of determination (R^2) for the decoded behavioral variables, averaged across behavioral variables within a behavioral group. For example, R^2 for x-velocity and y-velocity was averaged and reported as 'velocity R^2'. When computing R^2 for phase, circular distances and circular means were substituted for traditional distances and means in the R^2 definition. For the neural networks, the training/testing procedure was repeated 10 times with different random seeds. Reported performance for each behavioral group is the average performance across these 10 runs. Decoding R^2 was always computed at 5 ms resolution (set to match the resolution used by the Neural Latents Benchmark).

Decoder comparison

MINT was compared to four other decode algorithms: Kalman filter, Wiener filter, feedforward neural network, and a recurrent neural network (GRU). All decoders were provided with the same training/testing data and used the same evaluation epochs. Complete details on their implementations are provided in the Appendix.

Neuron dropping

The neuron dropping analyses in Figure 6 were performed on the MC_Maze-L dataset, which has 162 neurons. The 'retrained' neuron dropping analysis consisted of training and testing the decoder with reduced sets of neurons. The 'zeroed' neuron dropping analysis consisted of training and testing the decoder with all 162 neurons, but in the test set a subset of those neurons were artificially set to never spike. The analyses were performed for the following neuron counts: $[1, 2, 3, 4, 5, 10, 15, \ldots, 150, 155, 160, 162]$. At each neuron count, the 'retrained' and 'zeroed' analyses were each repeated 50 times (with different neurons randomly dropped at each repetition). Linear interpolation between these neuron counts was then applied to generate R^2 values for every neuron count between 1 and 162.

MINT variants

The analysis in Figure S3a-c required training and testing several variations on MINT for Area2_Bump, MC_Cycle, MC_Maze, and MC_RTT. These variations are described here.

The first variation determines how behavior is decoded from the neural state estimate. The 'Direct MINT Readout' utilizes the direct association between neural and behavioral states that is standard for MINT. The 'Linear MINT Readout' begins by estimating the neural state using MINT, but behavior is then decoded as a weighted sum of the rates comprising the neural state estimate. The weights are learned from training data using ridge regression, with the L2 regularization term in the regression optimized via a grid search with 5-fold cross validation. To improve the quality of the fit, a lag between neural and behavioral states was included (lags set to match the values used in Pei et al. [77]; MC_Cycle set to 100 ms).

The second variation determined whether interpolation was used. When interpolation was used, the same methodology used for Figure 4 analyses was followed (6 candidate states for MC_RTT; 2 candidate states for other datasets). When interpolation was not used, the neural state was selected from the library of neural trajectories as the state that maximized the log-likelihood of the observed spikes.

The third variation determined whether decoding occurred causally or acausally. When decoding causally, the spiking observations all came prior to the decoded moment. When decoding acausally, the spiking observations were centered on the decoding moment. For the analysis in Figure S3a-c, the extent of spiking observations (window length) was optimized separately for causal vs. acausal decoding to give each variant the opportunity to perform as well as possible. The causal window lengths are reported in Table 4. The acausal window lengths were 560 ms for Area2_Bump, 400 ms for MC_Cycle, 580 ms for MC_Maze, and 660 ms for MC_RTT.

SNR criteria for threshold crossings

When decoding based on threshold crossings in Figure S3d, a signal-to-noise ratio (SNR) was computed for each electrode channel. Then, only electrode channels with SNR > 2 were utilized for decoding. The SNR was computed as the ratio of the range of firing rates (determined from trial-averaged rates) and the standard deviation of the firing rate residuals. The firing rate residuals on each trial were computed as the difference between the filtered spikes and the corresponding trial-averaged rates. If the standard deviation was less than 5 spikes/second, it was set to 5 spikes/second in the SNR computation. This ensured that channels with low variability in the residuals due to saturation at small or large firing rates were rejected. For example, an electrode channel that almost never spikes will have very low variability in the residuals simply because the firing rate is typically 0.

Neural Latents Benchmark

All neural state estimation results in Figure 5 came from the Neural Latents Benchmark (`https://neurallatents.github.io/`), a neural latent variable modeling competition released through the 2021 NeurIPS Datasets & Benchmarks Track [77]. For each of 7 datasets, the benchmark provided training data containing simultaneous spiking activity and behavioral measurements. The neurons in the training data were divided into two sets: held-in neurons and held-out neurons. For the test data, the spiking activity of the held-in neurons was provided, but the spiking activity of the held-out neurons and the behavioral data was kept private by the benchmark curators. These splits are provided in Table 5. The evaluation epochs used are the same as those used in Table 4. (For DMFC_RSG, the evaluation epoch was the 1500 ms leading up to the 'go' time.) In the test data, the held-in spiking activity was only provided for time points within the evaluation epochs. For each dataset, submissions consisted of rate estimates for every trial in the training and test sets (held-in rates and held-out rates). The benchmark was fundamentally acausal: rate estimates at a given moment could leverage spiking activity from the entire trial. The data in Figure 5 reflect all benchmark submissions up through Feb. 24th, 2023.

MINT submissions

MINT first learned idealized neural trajectories during training as described in Learning idealized trajectories. Then, the neural trajectories were partitioned by neurons into two libraries: a held-in library of trajectories and a held-out library of trajectories. The held-in library contained neural states for the held-in neurons only. The held-out library similarly contained neural states for the held-out neurons only. MINT was run using the held-in library of trajectories to estimate a neural state in the held-in neural state space. A direct association between held-in neural states and held-out neural states (the same direct association typically used to map neural states to behavioral states) was then used generate rate estimates for the held-out neurons. Rate estimates were saturated such that they could not be lower than 0.1 spikes/second. Interpolation was utilized across indices and conditions (with modifications for DMFC_RSG and MC_RTT as described in Other forms of interpolation).

The window length used for acausal neural state estimation on each dataset is provided in Table 5. Note that test set data was not provided outside of the evaluation epochs. Thus, despite estimating rates acausally, the spiking observations often could not be centered on the estimated time. For example, when estimating the neural state for Area2_Bump, the window length was 500 ms, but the held-in spiking activity only spanned a 600 ms period. Thus, only neural states in the 250-350 ms portion of this 600 ms period could be estimated based on 500 ms of centered spiking activity. The neural states outside of this 250-350 ms zone were estimated by either propagating the earliest estimate backward or the latest estimate forward. This propagation is possible because each neural state estimate either occurs on a trajectory with a unique past and future or is an interpolation between trajectories that each have a unique past and future (i.e. the interpolation parameters are frozen and the states being interpolated are propagated backward or forward along their trajectories).

In addition to requiring test set rate estimates, the benchmark required training set rate estimates (these were needed for the velocity R^2 metric, see Evaluation metrics). Each moment in each trial of the training data corresponded to a particular condition c and a time within the execution of that condition k (with the exception of MC_RTT, which will be discussed in a moment). Thus, training rate estimates were simply constructed by assigning each moment within each training trial a vector of rates $\overline{\mathbf{x}}_k^c$ for the corresponding c and k. For MC_RTT, the training rate estimates were simply the single-trial rates learned via AutoLFADS.

Dataset	Training trials	Test trials	Held-in neurons	Held-out neurons	Window length (ms)
Area2_Bump	364	98	49	16	500
DMFC_RSG	1006	283	40	14	1500
MC_Maze	2295	574	137	45	500
MC_Maze-L	500	100	122	40	500
MC_Maze-M	250	100	114	38	500
MC_Maze-S	100	100	107	35	500
MC_RTT	1080	271	98	32	500

Table 5. Details for neural state estimation results. Note that the training trial counts match the total number of trials reported in Table 2. This reflects that the Neural Latents Benchmark utilized an additional set of test trials not reflected in the Table 2 trial counts. The test trials used for this analysis have ground truth behavior hidden by the benchmark creators and are therefore only suitable for this analysis.

Evaluation metrics

The evaluation metrics used in this paper are briefly described below. Detailed explanations can be found in Pei et al. [77].

Bits per spike was used to assess how well the rate estimates for the held-out neurons on the test data matched the actual held-out spiking activity. The metric is computed by first computing the log-likelihood of the observed spikes, given rate estimates, across all neurons. Then, the log-likelihood that would have been obtained by letting the rate estimates be the neurons' mean rates is subtracted off. Finally, the metric is normalized. Positive values indicate that the held-out rate estimates are more predictive of held-out spiking activity than the mean rates.

PSTH R^2 was computed by collecting all rate estimates on the test set, sorting them by condition, and averaging rate estimates across trials of the same condition. Then the R^2 was computed between these rate-estimate-derived PSTHs and the empirical PSTHs (computed by smoothing spikes and averaging across trials within conditions). Empirical PSTHs were computed using trials from both the training and test sets. A separate R^2 value was generated for each neuron — the reported values are the average R^2 across all neurons. This metric could not be computed for MC_RTT due to lack of condition structure.

Velocity R^2 was computed by regressing lagged x- and y-velocity of the hand against estimated rates in the training data (ridge regression). Then, the linear mapping learned via regression is applied to estimated rates in the test data to generate estimated x- and y-velocity of the hand. R^2 is computed between estimated velocities and actual velocities on the test data and averaged across the x- and y-components. This metric was not computed for DMFC_RSG because the evaluated portion of the trials did not involve motion of the monkey's arm.

Hyperparameter optimization

MINT has very few hyperparameters, all of which can be readily set by hand. These hyperparameters typically relate straightforwardly to properties of the task or data, and performance is robust to their exact values (Fig. S1). For example, we did not optimize bin size (Δ). Rather, we let $\Delta = 20$ ms for all analyses, reasoning that this value would reduce computation time while remaining considerably shorter than the timescale over which rates change. MINT's only other relevant hyperparameters are window length (duration of spiking history considered at each moment) and the number of candidate states to use for interpolation. The number of candidate states was simply set to two for all datasets except MC_RTT. For MC_RTT, the nature of the training data argued for considering more candidate states during the interpolation stage. Thus, the number of states was optimized using a grid search with 10-fold cross validation on the training set (using velocity decoding R^2 as an objective function to maximize). Window length was also optimized via this grid search procedure, separately for each dataset. In some cases (e.g. the maze datasets), we still chose to standardize window length across datasets because optimization yielded similar values. Because neural state estimation was acausal and decoding analyses were causal, window length was also optimized separately across these analyses. For the simulated datasets, we chose not to optimize window length at all – the maze simulations and multitask spiking network were simply set to use the same window length as MC_Maze.

This approach to hyperparameter selection contrasts with our approach for the non-MINT decoders. As described in the Appendix, the non-MINT decoders utilized a more sophisticated technique to select hyperparameters: Bayesian optimization [75]. Additionally, the non-MINT decoders were allowed the flexibility of using a different window length for each behavioral group. The choice not to provide MINT with the same flexibility was not arbitrary. Rather, this choice emphasizes a particularly useful aspect of MINT: any relevant behavioral variable can be read out from the same neural state estimate. There is no need to use separate training or decoding procedures for different behavioral variables.

The methods we used for learning neural trajectories to use with MINT also contained hyperparameters (e.g.

Table 3). For all datasets involved in the Neural Latents Benchmark (except MC_RTT), these hyperparameters were optimized using the grid search procedure (using bits per spike as an objective function to maximize). These hyperparameters were then re-used for decoding analyses with no additional optimization. For MC_RTT, we utilized AutoLFADS to learn neural trajectories, which contains a built-in procedure for optimizing hyperparameters. MC_Cycle, the maze simulations, and the multitask network were not part of the Neural Latents Benchmark and therefore needed their trajectory-learning hyperparameters set in a different way. For MC_Cycle, these hyperparameters were optimized using the grid search procedure with velocity decoding R^2 as the objective function. The maze simulations simply used the same hyperparameters as MC_Maze. For the multitask network, the trajectory learning hyperparameters were set very conservatively by hand to only perform very light temporal smoothing on spikes along with standard trial-averaging.

The example decoding results in Fig. 3 and the Supp. Video used the same hyperparameters that were used in generating the quantitative decoding results in Fig. 4.

Supplementary Figures

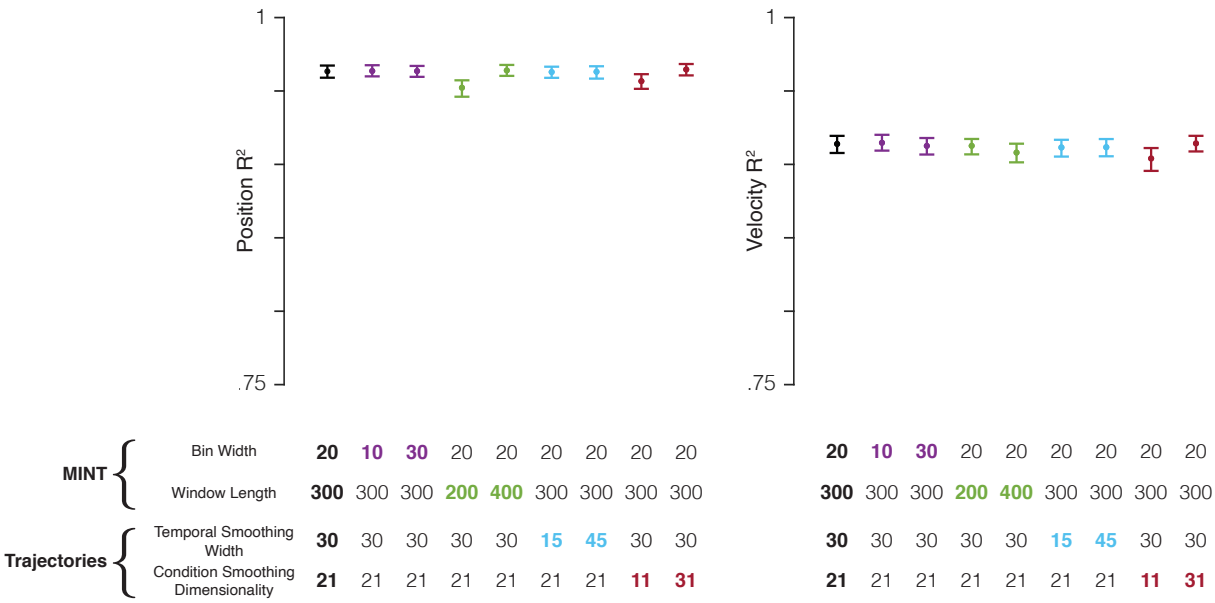

Figure S1. MINT's decoding performance is robust to the choice of hyperparameters. MINT was run on the MC_Maze dataset with systematic perturbations to two of MINT's hyperparameters: bin width (ms) and window length (ms). Bin width is the size of the bin in which spikes are counted: 20 ms for all analyses in the main figures. Window length is the length, in milliseconds, of spiking history that is considered. For all main analyses of the MC_Maze dataset, this was 300 ms, i.e. MINT considered the spike count in the present bin and in 14 previous bins. Perturbations were also made to two hyperparameters related to learning neural trajectories: temporal smoothing width (standard deviation of Gaussian filter applied to spikes) and condition-smoothing dimensionality (see Methods). These two hyperparameters describe how aggressively the trial-averaged data are smoothed (across time and conditions, respectively) when learning rates. Baseline decoding performance (*black circles*) was computed using the same hyperparameters that were used with the MC_Maze dataset in the analyses from Figure 3 and Figure 4. Then, decoding performance was computed by perturbing each of the four hyperparameters twice (*colored circles*): once to ∼50% of the hyperparameter's baseline value and once to ∼150%. Trials were bootstrapped (1000 resamples) to generate 95% confidence intervals (*error bars*). Perturbations of hyperparameters had little impact on performance. Altering bin width had essentially no impact, nor did altering temporal smoothing. Shortening window length had a negative impact, presumably because MINT had to estimate the neural state using fewer observations. However, the drop in performance was minimal: R^2 dropped by .011 for position decoding only. Reducing the number of dimensions used for across-condition smoothing, and consequently over-smoothing the data, had a negative impact on both position and velocity decoding. Yet again this was small: e.g. velocity R^2 dropped by .010. These results demonstrate that MINT can achieve high performance using hyperparameter values that span a large range. Thus, they do not need to be meticulously optimized to ensure good performance. In general, optimization may not be needed at all, as MINT's hyperparameters can often be set based on first principles. For example, in this study, bin width was never optimized either for specific datasets or in general. We chose to always count spikes in 20 ms bins (except in the perturbations shown here) because this duration is long enough to reduce computation time yet short relative to the timescales over which rates change. Additionally, window length can be optimized (as we did for decoding analyses), but it could also simply be chosen to roughly match the timescale over which past behavior tends to predict future behavior. Temporal smoothing of trajectories when building the library can simply use the same values commonly used when analyzing such data. For example, in prior studies, we have used smoothing kernels of width 20 to 30 ms when computing trial-averaged rates, and these values also support excellent decoding. Condition smoothing is optional and need not be applied at all, but may be useful if one wishes to record fewer trials for more conditions. For example, rather than record 15 trials for 8 reach directions, one might wish to record 5 trials for 24 conditions, then use condition smoothing to reduce sampling error.

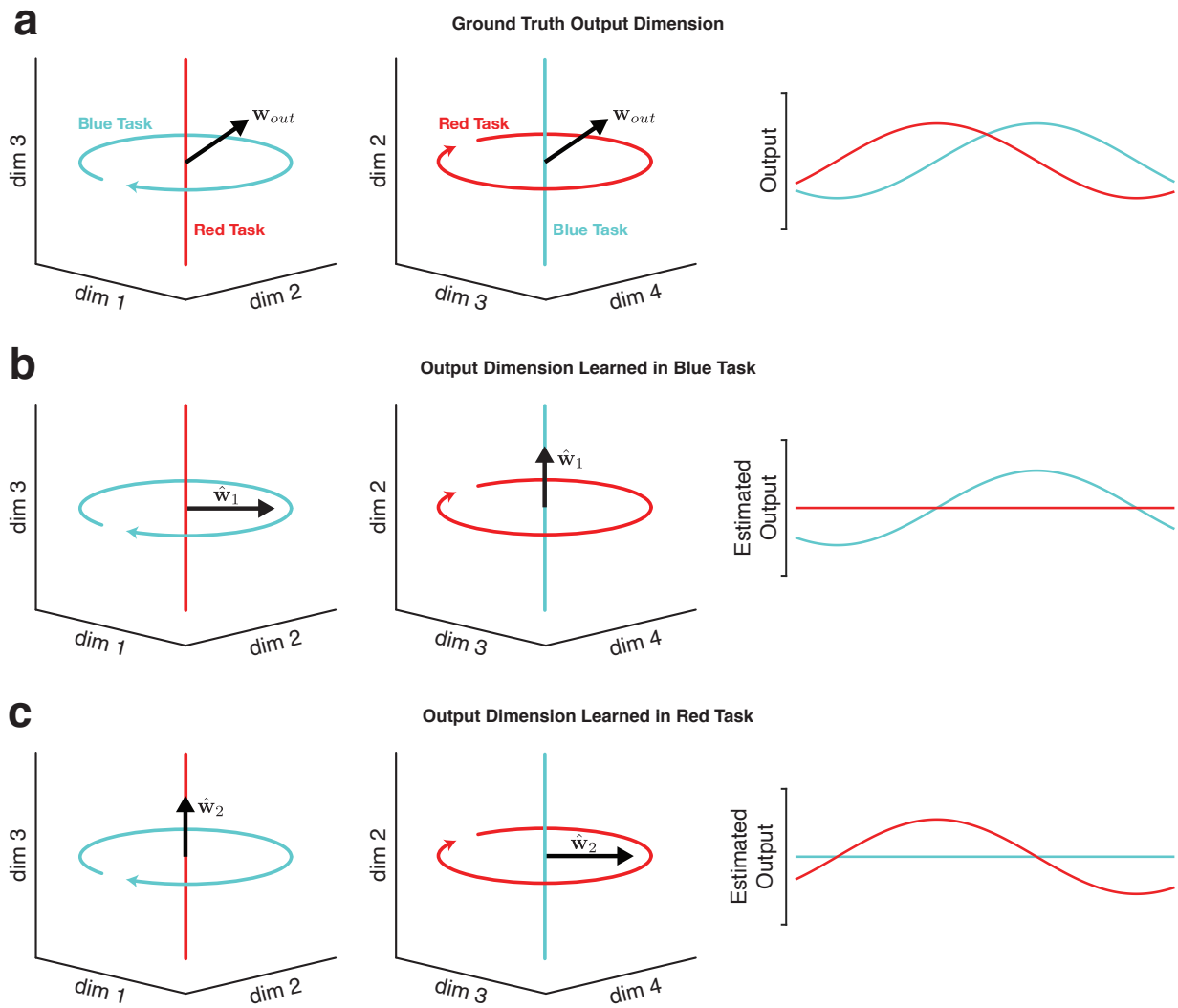

Figure S2. Illustration of why decoding will typically fail to generalize across tasks when neural trajectories occupy orthogonal subspaces. In theory, learning the output-potent dimensions in motor cortex would be an effective strategy for biomimetic decoding that generalizes to novel tasks. In practice, it is difficult (and often impossible) to statistically infer these dimensions without observing the subject's full behavioral repertoire (at which point generalization is no longer needed because all the relevant behaviors were directly observed). **a)** In this toy example, two neural trajectories occupy fully orthogonal subspaces. In the 'blue task', the neural trajectory occupies dimensions 1 and 2. In the 'red task', the neural trajectory occupies dimensions 3 and 4. Trajectories in both subspaces have a non-zero projection onto $\mathbf{w}_{out}$, enabling neural activity from each task to drive the appropriate output. The output at right is simply the projection of the blue-task and red-task trajectories onto $\mathbf{w}_{out}$. **b)** Illustration of the difficulty of inferring $\mathbf{w}_{out}$ from only one task. In this example, $\mathbf{w}_{out}$ is learned using data from the blue task, by linearly regressing the observed output against the neural trajectory for the blue task. The resulting estimate of $\hat{\mathbf{w}}_1$ correctly translates neural activity in the blue task into the appropriate time-varying output, but fails to generalize to the red task. This failure to generalize is a straightforward consequence of the fact that $\hat{\mathbf{w}}_1$ was learned from data that didn't explore dimensions 3 and 4. Note that this same phenomenon would occur if $\hat{\mathbf{w}}_1$ were estimated by regressing intended output versus neural activity (as might occur in a paralyzed patient) **c)** Estimating the output-potent dimension based only on the red task yields an estimate $\hat{\mathbf{w}}_2$ that fails to generalize to the blue task. This phenomenon is illustrated here for a linear readout, but would apply to most nonlinear methods as well, unless some other form of knowledge can allow interpretation of neural trajectories in previously unseen dimensions.

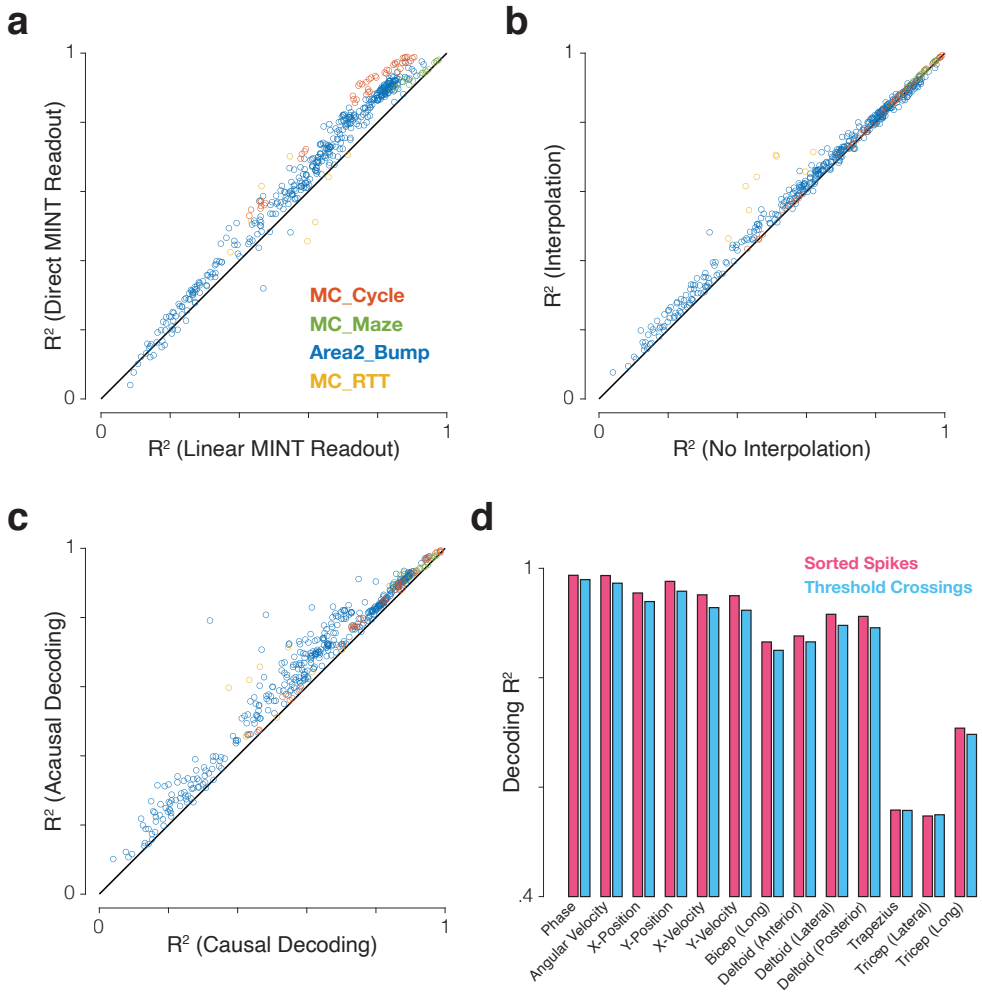

Figure S3. Impact of different modeling and preprocessing choices on performance of MINT. For most applications we anticipate MINT will employ direct decoding that leverages the correspondence between neural and behavioral trajectories, but one could also choose to linearly decode behavior from the estimated neural state. For most applications, we anticipate MINT will use interpolation amongst candidate neural states on neighboring trajectories, but one could also restrict decoding to states within the trajectory library. For real-time applications we anticipate MINT will be run causally, but acausal decoding (using both past and future spiking observations) could be used offline or even online by introducing a lag. We anticipate MINT may be used both in situations where spike events have been sorted based on neuron identity, and situations where decoding simply uses channel-specific unsorted threshold crossings. Panels **a-c** explore the first three choices. Performance was quantified for all 8 combinations of: direct MINT readout vs. linear MINT readout, interpolation vs. no interpolation, causal decoding vs. acausal decoding. This was done for 121 behavioral variables across 4 datasets for a total of 964 R^2 values. The 'phase' behavioral variable in MC_Cycle was excluded from 'linear MINT readout' variants because its circularity makes it a trivially poor target for linear decoding. **a)** MINT's direct neural-to-behavioral-state association outperforms a linear readout based on MINT's estimated neural state. Performance was significantly higher using the direct readout (ΔR^2 = .061 ± .002 SE; p<.001, paired t-test). Note that the linear readout still benefits from the ability of MINT to estimate the neural state using all neural dimensions, not just those that correlate with kinematics. **b)** Decoding with interpolation significantly outperformed decoding without interpolation (ΔR^2 = .018 ± .001 SE; p<.001, paired t-test). **c)** Running acausally significantly improved performance relative to causal decoding (ΔR^2 = .051 ± .002 SE; p<.001, paired t-test). Although causal decoding is required for real-time applications, this result suggests that (when tolerable) introducing a small decoding lag could improve performance. For example, a decoder using 200 ms of spiking activity could introduce a 50 ms lag such that the decode at time t is rendered by generating the best estimate of the behavioral state at time $t-50$ using spiking data from $t-200$ through t. **d)** Decoding performance for 13 behavioral variables in the MC_Cycle dataset when sorted spikes were used (112 neurons, *pink*) versus 'good' threshold crossings from electrodes for which the signal-to-noise ratio of the firing rates exceeded a threshold (93 electrodes, SNR > 2, *cyan*). The loss in performance when using threshold crossings was small (ΔR^2 = -.014 ± .002 SE). SE refers to standard error of the mean.

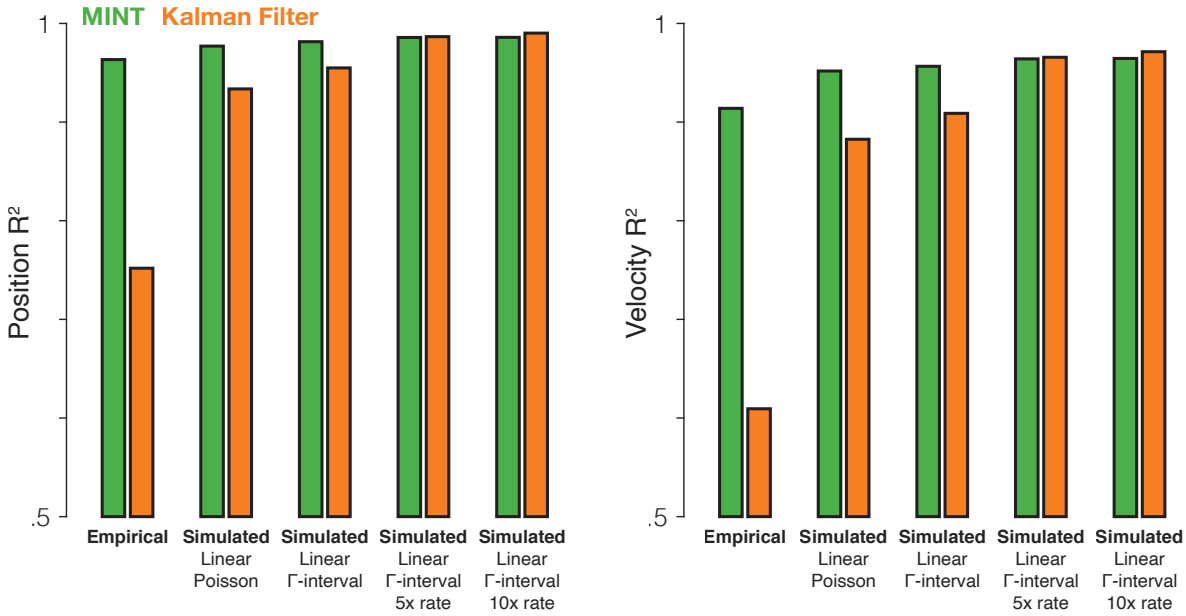

Figure S4. The Kalman filter's relative performance would improve if neural data had different statistical properties. We compared MINT to the Kalman filter across one empirical dataset (MC_Maze) and four simulated datasets (same behavior as MC_Maze, but simulated spikes). Simulated firing rates were linear functions of hand position, velocity, and acceleration (as is assumed by the Kalman filter). The means and standard deviations (across time and conditions) of the simulated firing rates were matched to actual neural data (with additional rate scaling in two cases). The Kalman filter assumes that observation noise is stationary and Gaussian. Although spiking variability cannot be Gaussian (spike counts must be positive integers), spiking variability can be made more stationary by letting that variability depend less on rate. Thus, although the first simulation generated spikes via a Poisson process, subsequent simulations utilized gamma-interval spiking (gamma distribution with $\alpha = 2$ and $\beta = 2\lambda$, where λ is firing rate). Gamma-interval spiking variability of this form is closer to stationary at higher rates. Thus, the third and fourth simulations scaled up firing rates to further push spiking variability into a more stationary regime at the expense of highly unrealistic rates (in the fourth simulation, a firing rate briefly exceeded 1800 Hz). Overall, as the simulated neural data better accorded with the assumptions of the Kalman filter, decoding performance for the Kalman filter improved. Interestingly, MINT continued to perform well even on the simulated data, likely because MINT can exploit a linear relationship between neural and behavioral states when the data argue for it and higher rates benefit both algorithms. These results demonstrate that algorithms like MINT and the Kalman filter are not intrinsically good or bad. Rather, they are best suited to data that match their assumptions. When simulated data approximate the assumptions of the Kalman filter, both methods perform similarly. However, MINT shows much higher performance for the empirical data, suggesting that its assumptions are a better match for the statistical properties of the data.

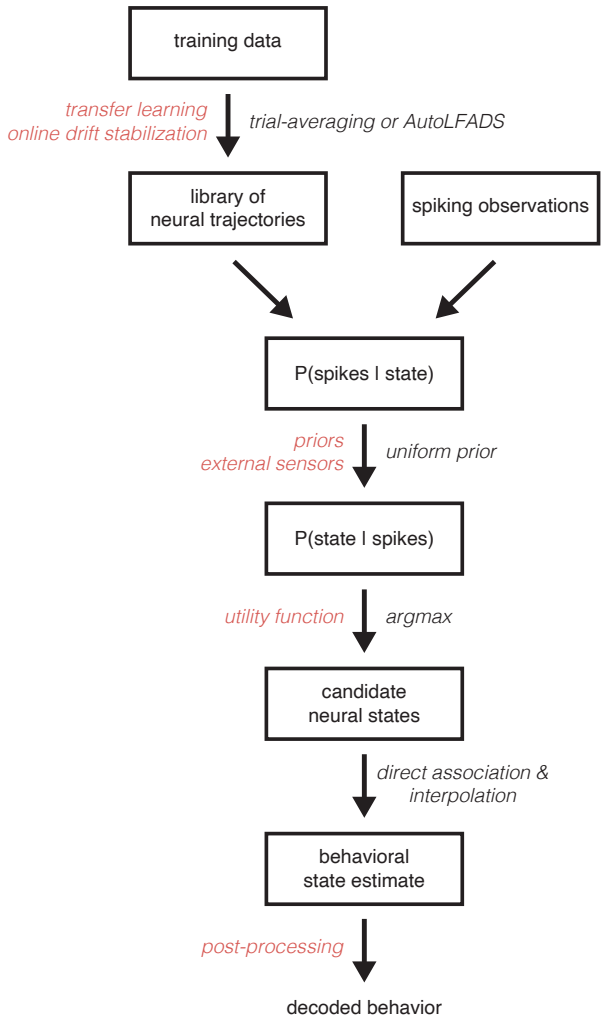

Figure S5. MINT is a modular algorithm for which a variety of modifications and extensions exist. The flowchart illustrates the standard MINT algorithm in *black* and lists potential changes to the algorithm in *red*. For example, the library of neural trajectories is typically learned via standard trial-averaging or a single-trial rate estimation technique like AutoLFADS. However, transfer learning could be utilized to learn the library of trajectories based on trajectories from a different subject or session. The trajectories could also be modified online while MINT is running to reflect changes in spiking activity that relate to recording instabilities. A potential modification to the method also occurs when likelihoods are converted into posterior probabilities. Typically, we assume a uniform prior over states in the library. However, that prior could be set to reflect the relative frequency of different behaviors and could even incorporate time-varying information from external sensors (e.g. state of a prosthetic limb, eye tracking, etc.) that include information about how probable each behavior is at a given moment. Another potential extension of MINT occurs at the stage where candidate neural states are selected. Typically, these states are selected based solely on spike count likelihoods. However, one could use a utility function that reflects a user's values (e.g. the user may dislike some decoding mistakes more than others) in conjunction with the likelihoods to maximize expected utility. Lastly, the behavioral estimate that MINT returns could be post-processed (e.g. temporally smoothed). MINT's modularity largely derives from the fact that the library of neural trajectories is finite. This assumption enables posterior probabilities to be directly computed for each state, rather than analytically derived. Thus, choices like how to learn the library of trajectories, which observation model to use (e.g. Poisson vs. generalized Poisson), and which (if any) state priors to use, can be made independently from one another. These choices all interact to impact performance. Yet they will not interact to impact tractability, as would have been the case if one analytically derived a continuous posterior probability distribution.

Supplementary Videos

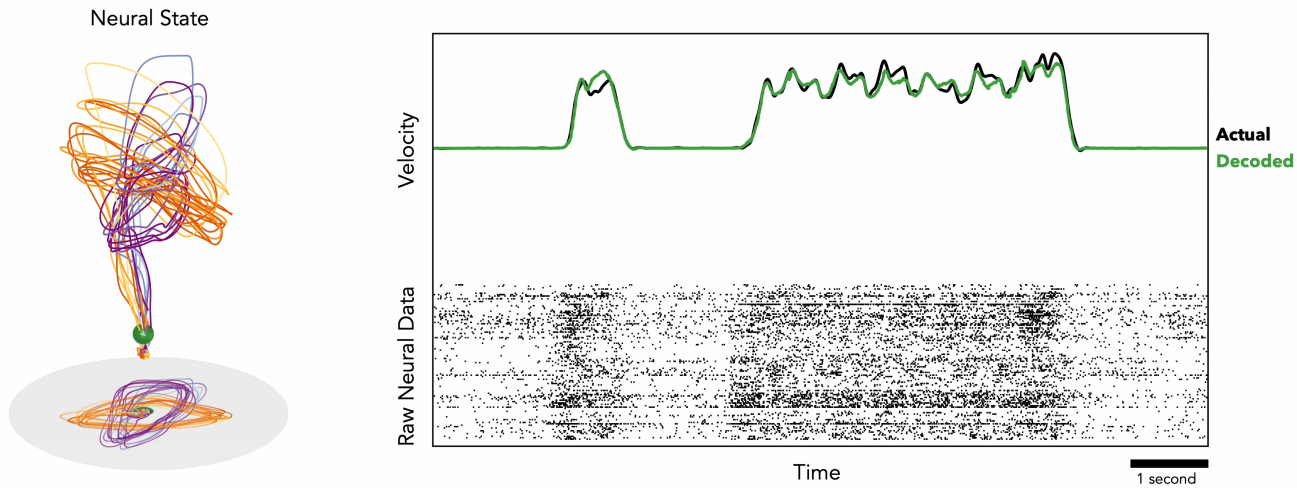

Figure SV1. Video demonstrating causal neural state estimation and behavioral decoding from MINT on the MC_Cycle dataset (see supplementary file). In this dataset, a monkey moved a hand pedal cyclically forward or backward in response to visual cues. The raster plot of spiking activity (112 neurons, *bottom right subpanel*) and the actual and decoded angular velocities of the pedal (*top right subpanel*) are animated with 10 seconds of trailing history. Decoding was causal; the decode of the present angular velocity (right hand edge of scrolling traces) was based only on present and past spiking. Cycling speed was $\sim$2 Hz. The underlying neural state estimate (*green sphere in left subpanel*) is plotted in a 3D neural subspace, with a 2D projection below. The present neural state is superimposed on top of the library of 8 neural trajectories used by MINT. The state estimate always remained on or near (via state interpolation) the neural trajectories. *Purple* and *orange* trajectories correspond to forward and backward pedaling conditions, respectively. The lighter-to-darker color gradients differentiate between trajectories corresponding to 1-, 2-, 4-, and 7-cycle conditions for each pedaling direction. Neural state estimate corresponds in time to the right edge of the scrolling raster/velocity plot. The three-dimensional neural subspace was hand-selected to capture a large amount of neural variance (59.6%; close to the 62.9% captured by the top 3 PCs) while highlighting the dominant translational and rotational structure in the trajectories.

Appendix: Decoder implementations

All decoders described in this section were provided with the same training and testing data as MINT. When decoding, spiking activity was binned every 20 ms for each neuron. Each decoder was given access to recent binned spike counts to use for decoding. The following sections will: 1) provide an overview of the notation and definitions needed to understand the subsequent equations (notation *will not* in general match the notation used for MINT), 2) describe the hyperparameter optimization technique used for all decoders in this appendix, and 3) describe each decoder's implementation in detail.

Definitions

Time Bins

Each decoder received binned spiking activity and decoded behavior at time-bin resolution. These time bins are indexed by k. When higher-resolution estimates were required (e.g. for evaluating decoding performance), the decoded variables were simply upsampled with a zero-order-hold.

Observations

The vector of spike counts at time bin k is denoted by the column vector $\mathbf{x}_k \in \mathbb{R}^N$ where N is the number of neurons. When decoding, each decoder was given access to the current time bin of spike counts and κ previous time bins of spike counts. κ was not a fixed value and varied depending on the method, dataset, and behavioral group being decoded.

Target Variables

The vector of behavioral variables at time bin k is denoted by the column vector $\mathbf{y}_k \in \mathbb{R}^M$ where M is the number of behavioral variables. This corresponds to the values of the behavioral variables at the end of the time bin.

Decoded Variables

The vector of decoded behavioral variables at time bin k is denoted $\hat{\mathbf{y}}_k$.

Training Data

Observations and target variables at time bin i of trial j in the training data are denoted $\mathbf{x}_i^{(j)}$ and $\mathbf{y}_i^{(j)}$.

Hyperparameter Optimization

The hyperparameters for each decoder were set using Bayesian optimization [75]. The training set was split into a reduced training set (80% of the trials from the full training set) and a validation set (the remaining 20% of trials). The method is provided with a range of hyperparameter values to search. The method then seeks to learn a set of hyperparameters that, when trained on the reduced training set, will lead to maximal decoding R^2 on the validation set. The optimization occurs via an iterative process that involves exploring the space of hyperparameters and exploiting knowledge of how certain hyperparameter sets performed in previous iterations. In all cases, we set the method to initially perform 10 iterations of random exploration of hyperparameter space, followed by 10 iterations of Bayesian optimization. The optimization was performed separately for each behavioral group (e.g. position vs. velocity) within each dataset. In the decoder-specific sections below, the exact hyperparameter values learned for each dataset and behavioral group are reported along with the hyperparameter ranges that were searched.

Wiener Filter

The Wiener filter [1] uses a model

$$\mathbf{y}_k = b + \sum_{i=0}^{\kappa} W_i \mathbf{x}_{k-i} + \boldsymbol{\epsilon}_k \tag{S1}$$

where $b \in \mathbb{R}^M$, $W_i \in \mathbb{R}^{M \times N}$, and the residuals are $\boldsymbol{\epsilon}_k \in \mathbb{R}^M$. Decoding occurs via the equation

$$\hat{\mathbf{y}}_k = W \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_{k-1} \\ \vdots \\ \mathbf{x}_{k-\kappa} \\ 1 \end{bmatrix} \tag{S2}$$

for some matrix W that minimizes the squared error of the residuals in the training data.

Parameter fitting:

For each trial j in the training data, we construct matrices

$$\mathbf{X}^{(j)} = \begin{bmatrix} \mathbf{x}_{\kappa+1}^{(j)} & \mathbf{x}_{\kappa+2}^{(j)} & \dots & \mathbf{x}_{T_j}^{(j)} \\ \mathbf{x}_{\kappa}^{(j)} & \mathbf{x}_{\kappa+1}^{(j)} & \dots & \mathbf{x}_{T_j-1}^{(j)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{1}^{(j)} & \mathbf{x}_{2}^{(j)} & \dots & \mathbf{x}_{T_j-\kappa}^{(j)} \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{S3}$$

$$\mathbf{Y}^{(j)} = \begin{bmatrix} \mathbf{y}_{\kappa+1}^{(j)} & \mathbf{y}_{\kappa+2}^{(j)} & \cdots & \mathbf{y}_{T_j}^{(j)} \end{bmatrix} \tag{S4}$$

where T_j is the number of time bins in trial j. Then, we concatenate across J trials to get

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}^{(1)} & \mathbf{X}^{(2)} & \dots & \mathbf{X}^{(J)} \end{bmatrix} \tag{S5}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{Y}^{(1)} & \mathbf{Y}^{(2)} & \dots & \mathbf{Y}^{(J)} \end{bmatrix} \tag{S6}$$

W can then be fit via a regularized regression,

$$W = \mathbf{Y}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top + \lambda I)^{-1} \tag{S7}$$

Dataset	Behavioral Group	Window Length (ms)	L2 Regularization (λ)
Area2_Bump	Position	560	350
	Velocity	320	320
	Force	600	900
	Joint Angles	300	250
	Joint Velocities	400	1410
	Muscle Lengths	380	0
	Muscle Velocities	460	940
MC_Cycle	Phase	1000	1730
	Angular Velocity	960	1470
	Position	920	1710
	Velocity	420	600
	EMG	560	1990
MC_Maze	Position	660	530
	Velocity	540	210
MC_Maze-L	Position	540	510
	Velocity	540	440
MC_Maze-M	Position	660	440
	Velocity	480	310
MC_Maze-S	Position	680	130
	Velocity	420	270
MC_RTT	Velocity	1180	1610

Table 6. Hyperparameters used for the Wiener filter. The L2 regularization term λ was optimized in the range $[0, 2000]$, with the optimized values rounded to the closest multiple of 10. Window lengths were optimized (in 20 ms increments) in the range $[200, 600]$ for Area2_Bump, $[200, 1000]$ for MC_Cycle, $[200, 1200]$ for MC_RTT, and $[200, 700]$ for MC_Maze, MC_Maze-L, MC_Maze-M, and MC_Maze-S. These ranges were determined by the structure of each dataset (e.g. Area2_Bump couldn't look back more than 600 ms from the beginning of the evaluation epoch without entering the previous trial). Window lengths are directly related to κ via Δ (e.g. $\kappa = 14$ would correspond to a window length of $\Delta(\kappa+1) = 20(14+1) = 300$ ms.)

Dataset	Behavioral Group	Lag (bins)
Area2_Bump	Position & Velocity	0
MC_Cycle	Position & Velocity	2
MC_Maze	Position & Velocity	4
MC_Maze-L	Position & Velocity	4
MC_Maze-M	Position & Velocity	4
MC_Maze-S	Position & Velocity	4
MC_RTT	Position & Velocity	2

Table 7. Hyperparameters used for the Kalman filter. The lag (in increments of 20 ms time bins) between neural activity and behavior was optimized in the range $[2, 8]$, corresponding to 40-160 ms, for all datasets except Area2_Bump. For Area2_Bump the lag was not optimized and was simply set to 0 due to the fact that, in a sensory area, movement precedes sensory feedback. Given that $\mathbf{x}_k$ aggregates spikes across the whole time bin, but $\mathbf{y}_k$ corresponds to the behavioral variables at the end of the time bin, the effective lag is actually half a bin (10 ms) longer — i.e. the effective range of lags considered for the non-sensory datasets was 50-170 ms.

Kalman filter

The Kalman filter [96] uses a model

$$\mathbf{y}_k = A\mathbf{y}_{k-1} + \mathbf{q}_k \tag{S8}$$
$$\mathbf{x}_k = C\mathbf{y}_k + \mathbf{r}_k \tag{S9}$$

where $\mathbf{q}_k \in \mathcal{N}(0, Q)$ and $\mathbf{r}_k \in \mathcal{N}(0, R)$. Decoding occurs via the standard update equations:

$$\hat{\mathbf{y}}_k^- = A\hat{\mathbf{y}}_{k-1} \tag{S10}$$
$$\hat{\mathbf{y}}_k = \hat{\mathbf{y}}_k^- + K_k(\mathbf{x}_k - C\hat{\mathbf{y}}_k^-) \tag{S11}$$

where

$$P_k^- = AP_{k-1}A^\top + Q \tag{S12}$$
$$K_k = P_k^- C^\top (CP_k^- C^\top + R)^{-1} \tag{S13}$$
$$P_k = P_k^- - K_k C P_k^- \tag{S14}$$

The state vector $\mathbf{y}_k$ includes 7 elements: position, velocity, and acceleration (x- and y-components of each) as well as a constant 1. The 1 was included to allow firing rates to have a constant offset. Acceleration was included as in [43] to improve the position and velocity estimates.

Parameter fitting:
For each trial j in the training set, we construct matrices

$$\mathbf{X}^{(j)} = \begin{bmatrix} \mathbf{x}_1^{(j)} & \mathbf{x}_2^{(j)} & \dots & \mathbf{x}_{T_j - lag}^{(j)} \end{bmatrix} \tag{S15}$$

$$\mathbf{Y}^{(j)} = \begin{bmatrix} \mathbf{y}_{1+lag}^{(j)} & \mathbf{y}_{2+lag}^{(j)} & \dots & \mathbf{y}_{T_j}^{(j)} \end{bmatrix} \tag{S16}$$

$$\mathbf{Y}_1^{(j)} = \begin{bmatrix} \mathbf{y}_{1+lag}^{(j)} & \mathbf{y}_{2+lag}^{(j)} & \dots & \mathbf{y}_{T_j - 1}^{(j)} \end{bmatrix} \tag{S17}$$

$$\mathbf{Y}_2^{(j)} = \begin{bmatrix} \mathbf{y}_{2+lag}^{(j)} & \mathbf{y}_{3+lag}^{(j)} & \dots & \mathbf{y}_{T_j}^{(j)} \end{bmatrix} \tag{S18}$$

where T_j is the number of time bins in trial j. Then, we concatenate across J trials to get

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}^{(1)} & \mathbf{X}^{(2)} & \dots & \mathbf{X}^{(J)} \end{bmatrix} \tag{S19}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{Y}^{(1)} & \mathbf{Y}^{(2)} & \dots & \mathbf{Y}^{(J)} \end{bmatrix} \tag{S20}$$

$$\mathbf{Y}_1 = \begin{bmatrix} \mathbf{Y}_1^{(1)} & \mathbf{Y}_1^{(2)} & \dots & \mathbf{Y}_1^{(J)} \end{bmatrix} \tag{S21}$$

$$\mathbf{Y}_2 = \begin{bmatrix} \mathbf{Y}_2^{(1)} & \mathbf{Y}_2^{(2)} & \dots & \mathbf{Y}_2^{(J)} \end{bmatrix} \tag{S22}$$

The parameters can then be fit as follows:

$$A = \mathbf{Y}_2 \mathbf{Y}_1^\top (\mathbf{Y}_1 \mathbf{Y}_1^\top)^{-1} \tag{S23}$$

$$C = \mathbf{X}\mathbf{Y}^\top (\mathbf{Y}\mathbf{Y}^\top)^{-1} \tag{S24}$$

$$Q = \frac{(\mathbf{Y}_2 - A\mathbf{Y}_1)(\mathbf{Y}_2 - A\mathbf{Y}_1)^\top}{T_1} \tag{S25}$$

$$R = \frac{(\mathbf{X} - C\mathbf{Y})(\mathbf{X} - C\mathbf{Y})^\top}{T} \tag{S26}$$

$$\hat{\mathbf{y}}_1^- = \frac{1}{J} \sum_{j=1}^{J} \mathbf{y}_{1+lag}^{(j)} \tag{S27}$$

$$P_1^- = \frac{1}{J} \sum_{j=1}^{J} (\mathbf{y}_{1+lag}^{(j)} - \hat{\mathbf{y}}_1^-)(\mathbf{y}_{1+lag}^{(j)} - \hat{\mathbf{y}}_1^-)^\top \tag{S28}$$

where T_1 and T are the number of columns in $\mathbf{Y}_1$ and $\mathbf{Y}$, respectively.

Feedforward Neural Network

The feedforward neural network [97] we use takes spike count observations (current and previous time bins) and flattens them into a long vector,

$$\mathbf{f}_k = \begin{bmatrix} \mathbf{x}_k^\top & \mathbf{x}_{k-1}^\top & \dots & \mathbf{x}_{k-\kappa}^\top \end{bmatrix}^\top \tag{S29}$$

centers and normalizes that vector,

$$\tilde{\mathbf{f}}_k = (\mathbf{f}_k - \mu) \oslash \sigma \tag{S30}$$

and then feeds it through multiple hidden network layers,

$$\mathbf{h}_k^1 = \mathrm{ReLU}(W_1 \tilde{\mathbf{f}}_k + b_1) \tag{S31}$$

$$\mathbf{h}_k^l = \mathrm{ReLU}(W_l \mathbf{h}_k^{l-1} + b_l),\ l \in \{2, \dots, L\} \tag{S32}$$

before finally reading out behavior,

$$\hat{\mathbf{y}}_k = W_{out} \mathbf{h}_k^L + b_{out} \tag{S33}$$

where $\mathbf{h}_k^l \in \mathbb{R}^D$, D is the number of units per hidden layer, and L is the number of hidden layers.

Parameter fitting:
For each trial j in the training set, we create flattened observations

$$\mathbf{f}_i^{(j)} = \begin{bmatrix} (\mathbf{x}_i^{(j)})^\top & (\mathbf{x}_{i-1}^{(j)})^\top & \dots & (\mathbf{x}_{i-\kappa}^{(j)})^\top \end{bmatrix}^\top \tag{S34}$$

for all $i > \kappa$ (sufficient spike count history doesn't exist for $i \leq \kappa$). We then let μ and σ be the element-wise mean and standard deviation of $\mathbf{f}_i^{(j)}$ across all observations in the training set (i.e. across all time bins in all J trials for which $i > \kappa$). The parameters W_{out}, b_{out}, W_l, and b_l for $l \in \{1, \dots, L\}$ are then learned by training the network with the Adam optimization routine [98] and a mean-squared error loss function. Training utilized dropout on the outputs of each hidden layer.

Dataset	Behavioral Group	Window Length (ms)	Hidden Layers (L)	Units/Layer (D)	Dropout Rate	Epochs
Area2_Bump	Position	440	4	690	0.20	58
	Velocity	520	5	260	0	62
	Force	240	4	560	0.01	54
	Joint Angles	320	7	480	0.15	59
	Joint Velocities	460	5	660	0.12	36
	Muscle Lengths	240	10	840	0.19	81
	Muscle Velocities	500	6	480	0.02	24
MC_Cycle	Phase	740	8	180	0.11	24
	Angular Velocity	700	8	710	0.01	43
	Position	360	9	470	0	45
	Velocity	460	5	330	0.25	55
	EMG	840	5	970	0.07	43
MC_Maze	Position	680	7	340	0.15	65
	Velocity	700	14	760	0	89
MC_Maze-L	Position	380	6	160	0.05	44
	Velocity	600	8	380	0.13	43
MC_Maze-M	Position	520	8	860	0.26	79
	Velocity	420	2	360	0.09	59
MC_Maze-S	Position	580	8	340	0.02	74
	Velocity	520	4	210	0.07	94
MC_RTT	Velocity	1040	2	700	0.15	17

Table 8. Hyperparameters used for the feedforward neural network. The number of hidden layers (L) was optimized in the range $[1, 15]$. The number of units per hidden layer (D) was optimized in the range $[50, 1000]$, with the optimized values rounded to the closest multiple of 10. The dropout rate was optimized in the range $[0, 0.5]$ and the number of training epochs was optimized in the range $[2, 100]$. Window lengths were optimized (in 20 ms increments) in the range $[200, 600]$ for Area2_Bump, $[200, 1000]$ for MC_Cycle, $[200, 1200]$ for MC_RTT, and $[200, 700]$ for MC_Maze, MC_Maze-L, MC_Maze-M, and MC_Maze-S.

GRU

The GRU neural network [99] we use takes spike count observations (current and previous time bins), centers and normalizes those observations,

$$\tilde{\mathbf{x}}_i = (\mathbf{x}_i - \mu) \oslash \sigma, \; i \in \{k-\kappa, \ldots, k\} \tag{S35}$$

and then feeds the observations sequentially into the network (initializing with $\mathbf{h}_{k-\kappa-1} = \mathbf{0}$),

$$\mathbf{z}_i = \text{sigmoid}(W_z \tilde{\mathbf{x}}_i + U_z \mathbf{h}_{i-1} + b_z) \tag{S36}$$

$$\mathbf{r}_i = \text{sigmoid}(W_r \tilde{\mathbf{x}}_i + U_r \mathbf{h}_{i-1} + b_r) \tag{S37}$$

$$\hat{\mathbf{h}}_i = \tanh(W_h \tilde{\mathbf{x}}_i + U_h (\mathbf{r}_i \odot \mathbf{h}_{i-1}) + b_h) \tag{S38}$$

$$\mathbf{h}_i = \mathbf{z}_i \odot \mathbf{h}_{i-1} + (1 - \mathbf{z}_i) \odot \hat{\mathbf{h}}_i \tag{S39}$$

ultimately reading out behavior from the final state,

$$\hat{\mathbf{y}}_k = W_{out} \mathbf{h}_k + b_{out} \tag{S40}$$

where $\mathbf{z}_i \in \mathbb{R}^D$, $\mathbf{r}_i \in \mathbb{R}^D$, $\hat{\mathbf{h}}_i \in \mathbb{R}^D$, $\mathbf{h}_i \in \mathbb{R}^D$, and D is the number of GRU units. The GRU hidden states do not persist from one decoding time bin to the next. Rather, at each decoding time bin, the GRU is re-initialized with $\mathbf{h}_{k-\kappa-1} = \mathbf{0}$ and run sequentially over recent history to generate an estimate of the behavior at the current time bin.

Parameter fitting:
μ and σ (both column vectors of length N) are computed as the mean and standard deviation, respectively, of the observed spike counts for each neuron in the training set. The parameters W_z, U_z, b_z, W_r, U_r, b_r, W_h, U_h, b_h, W_{out}, and b_{out} are then learned by training the network with the RMSProp optimization routine [100] and a mean-squared error loss function. Training utilized dropout both on the linear transformation of inputs and on the linear transformation of the recurrent state.

Dataset	Behavioral Group	Window Length (ms)	Units (D)	Dropout Rate	Epochs
Area2_Bump	Position	560	700	0.06	12
	Velocity	340	820	0.32	3
	Force	380	380	0.27	26
	Joint Angles	480	630	0.31	8
	Joint Velocities	260	390	0.27	9
	Muscle Lengths	460	990	0.34	45
	Muscle Velocities	420	870	0.19	47
MC_Cycle	Phase	460	580	0.18	49
	Angular Velocity	480	390	0.12	27
	Position	920	760	0.06	42
	Velocity	520	170	0.43	45
	EMG	840	250	0.40	36
MC_Maze	Position	640	740	0.27	4
	Velocity	520	800	0.34	11
MC_Maze-L	Position	620	640	0.40	49
	Velocity	580	420	0.36	31
MC_Maze-M	Position	620	820	0.41	48
	Velocity	660	840	0.29	30
MC_Maze-S	Position	460	310	0.39	27
	Velocity	680	500	0.44	45
MC_RTT	Velocity	540	890	0.40	8

Table 9. Hyperparameters used for the GRU network. The number of units (D) was optimized in the range $[50, 1000]$, with the optimized values rounded to the closest multiple of 10. The dropout rate was optimized in the range $[0, 0.5]$ and the number of training epochs was optimized in the range $[2, 50]$. Window lengths were optimized (in 20 ms increments) in the range $[200, 600]$ for Area2_Bump, $[200, 1000]$ for MC_Cycle, $[200, 1200]$ for MC_RTT, and $[200, 700]$ for MC_Maze, MC_Maze-L, MC_Maze-M, and MC_Maze-S.