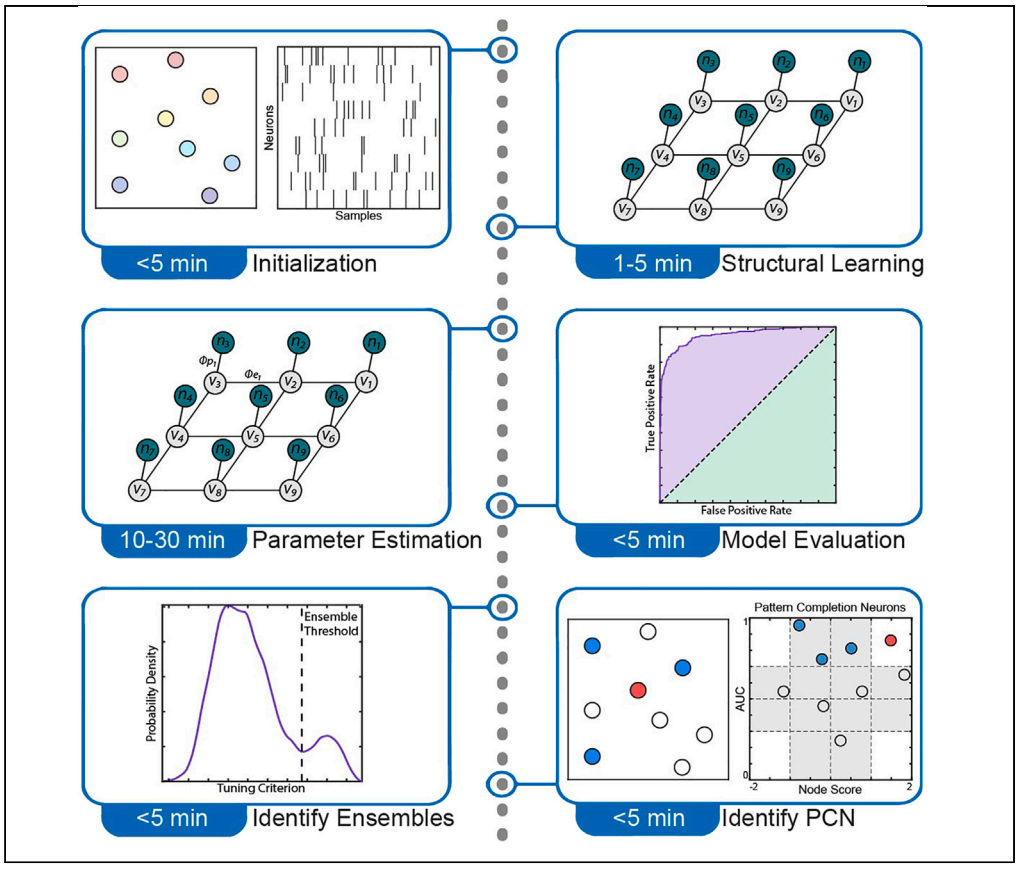
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Mapping neuronal ensembles and pattern-completion neurons through graphical models



Neuronal ensembles are coordinated groups of neurons that serve as functional building blocks of neural circuits. Here, we present PatMap, a computational toolbox for identifying pattern-completion neurons, key trigger cells capable of reactivating entire neuronal ensembles. We describe a protocol for modeling neural circuits as probabilistic graphical models, linking behavior with specific neuronal ensembles, and identifying their pattern-completion neurons. By linking the cellular and circuit level, PatMap provides a springboard for targeted manipulation and control of neural circuits.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Darik A. O'Neil,
Alejandro Akrouh,
Rafael Yuste

d.oneil@columbia.edu

Highlights

Model neural circuits as probabilistic graphical models

Decode user-defined features from neural activity

Identify embedded neuronal ensembles critical for model performance

Target pattern-completion neurons that reactivate entire neuronal ensembles

O'Neil et al., STAR Protocols 4, 102543
September 15, 2023
https://doi.org/10.1016/j.xpro.2023.102543

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Mapping neuronal ensembles and pattern-completion neurons through graphical models

Darik A. O'Neil,[1,2,3,*] Alejandro Akrouh,[1] and Rafael Yuste[1]

[1]Neurotechnology Center, Department of Biological Sciences, Columbia University, New York City, NY 10027, USA

[2]Technical contact

[3]Lead contact

*Correspondence: d.oneil@columbia.edu
https://doi.org/10.1016/j.xpro.2023.102543

SUMMARY

Neuronal ensembles are coordinated groups of neurons that serve as functional building blocks of neural circuits. Here, we present PatMap, a computational toolbox for identifying pattern-completion neurons, key trigger cells capable of reactivating entire neuronal ensembles. We describe a protocol for modeling neural circuits as probabilistic graphical models, linking behavior with specific neuronal ensembles, and identifying their pattern-completion neurons. By linking the cellular and circuit level, PatMap provides a springboard for targeted manipulation and control of neural circuits.

For complete details on the use and execution of this protocol, please refer to Carrillo-Reid et al. (2021).[1]

BEFORE YOU BEGIN

Large-scale recordings of populations of neurons have revealed that neural activity is characterized by rich, intricate, and often stereotyped dynamics.[2] In particular, small groups of neurons with coordinated activity, defined as neuronal ensembles, appear to represent a variety of sensory and cognitive states.[3] The objective of our computational toolbox for pattern completion mapping (PatMap) is to statistically model the neuronal interactions that underlie these ensembles and leverage these insights to identify pattern completion neurons—neurons whose activation is capable of triggering entire neuronal ensembles.[4] In our approach, pattern completion neurons are defined by their stimulus specificity and robust functional connectivity.[5] The identification of pattern completion neurons is accomplished through the use of probabilistic graphical models known as conditional random fields. A basic introduction to the concept of conditional random fields can be found here.[6] In a biological interpretation, each *node* in these models represents an individual neuron. These neuronal nodes are connected to other nodes through *edges*, each of which represent an inferred interaction between neurons (Figure 1A).

Mathematical primer

PatMap consists of three stages: learning the circuit's graphical structure, estimating its parameters, and evaluating the graphical model for pattern completion neurons. A detailed mathematical description follows.

We construct conditional random fields (CRFs) given a binary spike matrix of M samples and N neurons. We consider each neuron n within the dataset as a node within an undirected graphical model (Figure 1A). The conditional dependencies of these nodes are defined by their edges. The state of every node and edge in the graphical model is described by a feature vector of length M. Every m-th row of the matrix may be considered a population vector—an N-dimensional vector indicating the

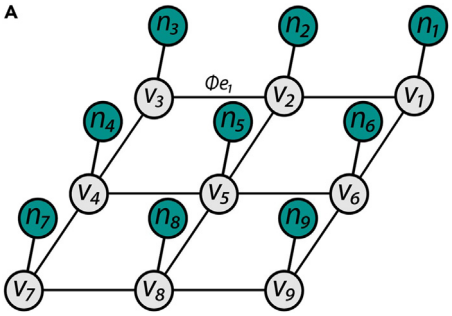


Figure 1. Conditional random field model of neural circuits
(A) Shaded nodes represent observed neurons. Each observed neuron is associated with an unshaded model node. The unshaded model nodes are connected by edges which describe the inferred functional interactions of the neuronal circuit. The nodes and edges of the model are described by scalar scores known as node (Φ_p) and edge potentials (Φ_e), respectively.

configuration of the circuit at a specific moment in time: $y = [y^1, y^2, \ldots, y^M]$, $y^m \in Y^N$. Given the feature vectors, $x = [x^1, x^2, \ldots, x^M]$, $x^m \in X^N$, the conditional probability of the population vector y^m given features x^m may be defined as:

$$p(y^m|x^m;\theta) = \frac{exp(\langle\varphi(x^m, y^m), \theta\rangle)}{Z(x^m, \theta)}$$

where φ is a vector of sufficient statistics in log-linear form, θ is a vector of parameters to be estimated, and Z is the partition function:

$$Z(x^m;\theta) = \sum_{y \in Y^N} exp(\langle(x^m, y), \theta\rangle)$$

Structural learning

The conditional probability can be factored over the described graphical structure $G = (V, E)$, where V is a set of neuronal nodes and E is a set of node-pairs connected through edges. That is, $E \subseteq V$.To learn this graphical structure (i.e., which neuronal nodes are connected by edges), we conduct ℓ_1-regularized neighborhood regression for each node v given vector of regression parameters ϑ and regularization parameter λ_s:[7]

$$\min_{\vartheta_v}\{\ell(\vartheta; x) + \lambda_s\|\vartheta_v\|_1$$

where

$$\ell(\vartheta; x) = \frac{-1}{k}\sum_{i=1}^{k} \log\frac{\exp\left(2x_v\sum_{t \in V_v}\vartheta_{vt}xt\right)}{\exp\left(2x_v\sum_{t \in V_v}\vartheta_{vt}xt\right)+1}$$

and

$$\vartheta_v = \{\vartheta_{vu}, u \in V\ v\}$$

To efficiently learn a set of λ_s for each node v, we minimize the following objective function for each node v using iteratively reweighted penalized least squares:[8]

$$\min_{(\beta_0,\beta) \in \mathbb{R}^{P+1}} -1\left[\frac{1}{V}\sum_{i=1}^{V} y_i\left(\beta_0 + x_i^T\beta\right) - \log\left(1 + e^{\left(\beta_0 + x_i^T\beta\right)}\right)\right] + \lambda_s\left[\frac{(1-\alpha)\|\beta\|_2^2}{2} + \alpha\|\beta\|_1\right]$$

where α controls the balance between ℓ_1 and ℓ_2 regularization. Given $\alpha = 1$, this objective function learns the mapping of the ℓ_1-regularized regression of node v and the nodes $V\ v$ given any λ_s. Once learned, this gives us the ability to determine the associated graphical structure for any λ_s rapidly.

Parameter estimation

Given a graphical structure $G = (V,E)$, it follows that the parameters θ for the nodes and edges can be written separately in an overcomplete representation. In this formulation, φ_V is a set of two vectors of sufficient statistics for two node possible node states ('0','1') with parameters θ_V, while φ_E is a

set of four vectors of sufficient statistics for the four possible edge states ('00', '01', '10', '11') with parameters θ_E. That is,

$$p(Y|X;\theta) = \frac{\exp\left(\sum_{i\in V}\varphi_i(X,Y_i)\theta_i+\sum_{j\in E}\varphi_j(X,Y_j)\theta_j\right)}{Z(X;\theta)}$$

Given the M observed population vectors $\{y^1,y^2,\ldots,y^M\}$, and corresponding features vectors $\{x^1,x^2,\ldots,x^M\}$, we learn θ by maximizing the ℓ_2-regularized log-likelihood of the observations:[9,10]

$$\ell(\theta;X,Y) = \sum_{k=1}^{S}\ell(\theta;X_k,Y_k) - \frac{\lambda''}{2}\|\theta\|^2$$

where the log-likelihood for each observation k is:

$$\ell(\theta;X_k,Y_k) = \langle\varphi(X_k,Y_k),\theta\rangle - logZ(X_k)$$

Hyperparameter optimization

To determine the optimal values of the hyperparameters λ_s and λ_p, we invoke a sequential model-based Bayesian optimization in which we maximize the objective:[11]

$$\max_{s\in\lambda_s,p\in\lambda_p} f(s,p)$$

over a set of withheld samples t where:

$$f(s,p) = \overline{\ell}(\theta;X_t,Y_t)$$

Here, we construct a model of this objective function as a Gaussian process. Specifically, we formulate a model relating the values of hyperparameters λ_s and λ_p with performance on a withheld validation dataset. We then iteratively update this surrogate model by select new combinations of hyperparameters to evaluate and incorporate their performance into the model

Identifying pattern completion neurons

To integrate information about user-defined features, we merge binary vectors indicating each feature (e.g., a visual stimulus) into the spike matrix. Thereafter, the log-likelihood ratio of each population vector (or, network configuration) having come from the trained model had the behavioral node been active or inactive is calculated:

$$\ell_{i,1-0} = \left\{log\left(p_{i,1}^m\right) - \log\left(p_{i,0}^m\right)\right\}, m = 1,\ldots,M$$

where

$$p_{i,1}^m = p\left(y^m\middle|x_{V-\{i\}}^m, x_i^m = 1;\theta\right)$$

$$p_{i,0}^m = p\left(y^m\middle|x_{V-\{i\}}^m, x_i^m = 0;\theta\right)$$

This ratio represents the ability of the model to predict the user-defined feature given the configuration of network. By using this metric of prediction performance alongside the '11' (co-activity) edge scores, a core set of neurons whose activity is both highly predictive of the user-defined feature and robust connectivity—pattern completion neurons—maybe identified.

Institutional permissions

All experimental procedures used to generate sample data were carried out in accordance with the US National Institutes of Health and Columbia University Institutional Animal Care and Use Committee. To generate data for analysis in this software will require sufficient permissions from the relevant institutions.

Preparation

Timing: <10 min

Downloading PatMap

1. The PatMap toolbox can be downloaded from https://github.com/darikoneil/PatMap. The repository contains a collection of MATLAB functions, a graphical user interface (GUI), documentation (including a tutorial), an example dataset, and dependencies. The repository can also be downloaded directly using your operating system's terminal through git.

```
>git clone https://github.com/darikoneil/PatMap
```

2. After downloading, extract the PatMap zipped file and place the extracted folder in the location of your choosing.

Installing PatMap

3. PatMap does not require installation, but does require an installation of MATLAB. It is recommended that PatMap be run within MATLAB 2020b. Instructions on installing MATLAB can found here.

Launching PATMAP

4. Open MATLAB and enter the PatMap folder.
5. Enter PatMap in the command window to launch the GUI.

```
>app = pat_map();
```

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Excitatory neurons in visual cortex during drifting gratings	This article	https://github.com/darikoneil/PatMap/example_datasets/drifting_gratings
Excitatory neurons in hippocampal CA1 during contextual fear conditioning	This article	https://github.com/darikoneil/PatMap/example_datasets/contextual_fear
Excitatory neurons in the hippocampal CA1 during spatial navigation task	Krishnan et al.[12]	Krishnan, Seetha (2023) Hippocampus Reward Dataset (version 0.230416.2132) [data set]. DANDI archive. https://doi.org/10.48324/dandi.000462/0.230416.2132
Software and algorithms		
PatMap	Carrillo-Reid et al.[1]	https://github.com/darikoneil/PatMap/
MATLAB 2020b (recommended) or greater	MathWorks	https://www.mathworks.com/products/matlab.html
MATLAB Parallel Computing Toolbox (optional)	MathWorks	https://www.mathworks.com/products/parallel-computing.html
MATLAB Signal Processing Toolbox	MathWorks	https://www.mathworks.com/products/signal.html
MATLAB Statistics and Machine Learning Toolbox	MathWorks	https://www.mathworks.com/products/statistics.html
MATLAB Symbolic Math Toolbox	MathWorks	https://www.mathworks.com/products/symbolic.html

(*Continued on next page*)

Continued

REAGENT or RESOURCE	SOURCE	IDENTIFIER
MexCPP (modified)	Kui Tang	https://github.com/kuitang/mexcpp
QPBO v1.32 (modified)	Rother et al.[10]	https://pub.ista.ac.at/~vnk/software.html
GLMNet	Friedman et al., 2010[8]	https://glmnet.stanford.edu/articles/glmnet.htm
ParWaitBar	Olivier Trottier	https://github.com/oliviertrottier/parwaitbar
CmdLineProgressBar	Ita Katz	https://www.mathworks.com/matlabcentral/fileexchange/56871-command-line-progress-bar-waitbar
Other		
Tested operating system	Windows 10	https://www.microsoft.com/en-us/software-download/windows10
Tested operating system	Windows 11	https://www.microsoft.com/en-us/software-download/windows11
Tested operating system	Ubuntu 18.04 LTS	https://releases.ubuntu.com/
Tested operating system	Ubuntu 20.04 LTS	https://releases.ubuntu.com/
Tested operating system	Ubuntu 22.04 LTS	https://releases.ubuntu.com/
Memory	Scales with size of dataset (≥8 GB)	N/A

STEP-BY-STEP METHOD DETAILS

Here we describe step-by-step details in the analysis of an example dataset, from the importing of the dataset, to parameter selection, data validation, structural learning, parameter estimation, model selection, the identification and evaluation of neuronal ensembles, and the identification of pattern completion neurons. We have indicated the estimated time required for the example dataset. A comprehensive benchmarking can be found here to estimate the computation time for your own datasets.

Initialization

Timing: <5 min

In this portion of the protocol, we import our data, select dataset-specific parameters, and validate the compatibility of our dataset (Figure 2A). For this tutorial, we will utilize data collected from the calcium imaging of excitatory neurons in the visual cortex during the presentation of drifting gratings with 0°, 45°, 90°, and 135° orientations on an LCD screen. More information on this dataset can be found here.

Note: All relevant files can be found in the 'drifting_gratings' folder within 'example datasets'. The folder contains five files: 'visual_cortex_data.mat', 'visual_cortex_udf.mat', 'visual_cortex_rois.mat', 'visual_cortex_params.mat', 'metadata.txt'. The 'data' file contains an *M* sample by *N* neuron binary matrix where each element indicates whether the *n*-th neuron was spiking during the *m*-th sample. The 'udf' file contains two variables. The first variable 'udf' is an *M* sample vector of natural numbers indicating which drifting grating was presented during the *m*-th sample. The 'udf_labels' variable contains a cell array mapping the natural numbers in 'udf' and the presented drifting gratings. The 'rois' file is an optional file that contains the pixel masks for each neuron within the dataset. The parameters file is a structure containing the parameters for the analysis. The provided parameters file is simply the default parameters and is only provided to provide an example on loading a set of saved parameters. Finally, the 'metadata.txt' file contains a short description of the experiment.

⚠ CRITICAL: Imported data files must be an *M* sample by *N* neuron binary matrix where each value indicates the state of the neuron in the sample. Intuitively, the *state* of the neuron is a simple binary indication of spiking (i.e., active or inactive). Nevertheless, the matrix does

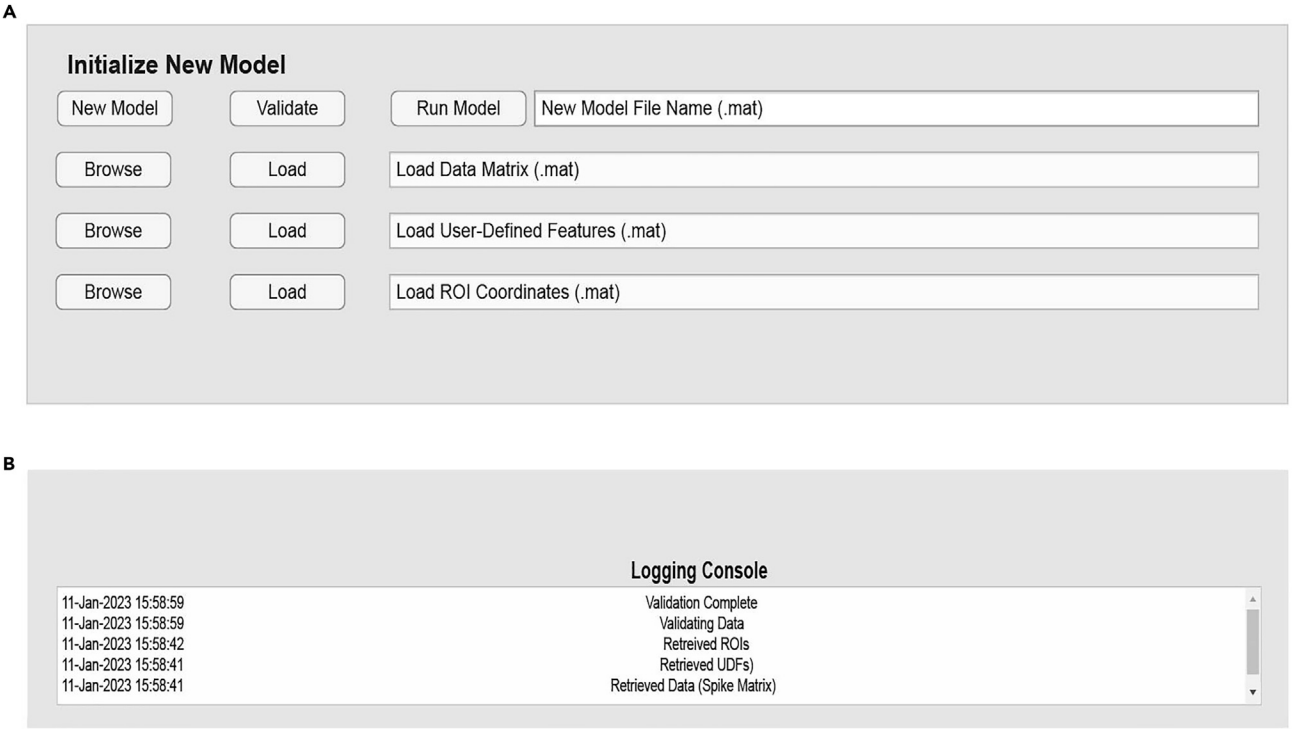


Figure 2. Importing data

(A) Import interface for loading and validating data. Note separate buttons for loading the spike matrix, user-defined features, and ROI coordinates.
(B) Logging console confirming successful import and validation of data.

not need to directly map to discrete spikes. Any binary measure of neural activity is sufficient. This 'data' matrix must be saved in the form of a '.mat' file (MATLAB binary file).

⚠ CRITICAL: Imported udf files must contain either an *M* sample vector of natural numbers where each element indicates the user-defined feature present in the *m*-th sample or an *M* sample by *N* user-defined feature binary matrix where each element indicates whether the *n*-th user-defined feature was present during the *m*-th sample. It is critical that each sample is directly matched with the samples of the dataset. An optional cell array of length *N* user-defined features may be provided that maps each user-defined feature with a text label.

⚠ CRITICAL: Imported ROIs must be properly formatted in the form of a '.mat' (MATLAB binary file). Acceptable formats include: (1) a 2D matrix in which the rows are neurons and the columns represent the center point in Cartesian coordinates with respect to the image (2) a 2D matrix in which the rows are neuron, the first two columns represent the center point in Cartesian coordinates with respect to the image, and the third column representing the imaging plane or (3) an exported Suite2P structure.

1. Importing your dataset.
   a. Press the browse button to open a pop-up file explorer.
   b. Locate your dataset within the file explorer and press the open button.

   Note: In this case, we will be selecting the 'visual_cortex_data.mat' file in the 'drifting_gratings folder located within 'example_datasets' folder.

   Note: The path to your file will be reflected in the associated textbox.

c. Click the load button to import your dataset.

Note: A preview of the dataset will be generated below.

2. Importing (potentially) encoded features, referred to as user-defined features (UDFs)
   a. Once again, press the browse button to open a pop-up file explorer.
   b. Locate your UDFs within the file explorer and press the open button.

   Note: In this case, we will be selecting the 'visual_cortex_udf.mat' file in the 'drifting_gratings' folder located within 'example_datasets' folder.

   Note: The path to your file will be reflected in the associated textbox.

   c. Click the load button to import the UDFs.
3. Optional: a file containing ROI coordinates can be imported for visual identification of ensemble neurons and pattern completion neurons, and superimposing the learned graphical structure onto the rois. If one is not provided, simulated rois will be generated.
   a. Press the browse button to open a pop-up file explorer.
   b. Locate your ROIs file within the file explorer and press the open button.

   Note: In this case, we will be selecting the 'visual_cortex_rois.mat' file in the 'drifting_gratings' folder located within 'example_datasets' folder.

   Note: The path to your file will be reflected in the associated textbox.

   c. Click the load button to import the ROIs. The ROIs will be plotted below.
4. Selecting Parameters.

   Note: The right panel displays the major parameters of the analysis pipeline. Relevant parameters are additionally displayed on the submenu of each step and are described as the user performs those steps later. These parameters are linked (i.e., changing one changes the other).

   Note: In this tutorial we will demarcate and outline parameter choices during the step they are relevant.

   a. Press the browse button to open a pop-up file explorer.
   b. Locate the parameters file within the file explorer and press the open button.

   Note: In this case, we will be selecting the 'visual_cortex_params.mat' file in the 'drifting_gratings' folder located within the 'example_datasets' folder.

   c. Click the load button to import the parameters. Do not worry if the parameters do not change. The supplied example parameters are simply the defaults.
5. Validating the Dataset.
   a. First, we specify the three parameters relevant for this stage.
      i. 'Shuffle Data' indicates whether to shuffle the dataset before learning. Shuffling the data here mitigates the influence of potential longitudinal changes in the dataset (e.g., unstable or dynamic representation).

   Note: It may seem that one would always wish to shuffle the dataset, however, there are situations where this is not the case. For example, one might wish to train a model on some behaviorally relevant portion of the dataset and consider whether it generalizes to a different portion. For example, before and after extinction of a conditioned stimulus.

ii. 'Training-Testing Split' indicates the fraction of samples used for training the graphical model. To mitigate overfitting of the model, a subset of the data should always be withheld at this stage (i.e., the *testing* data).
iii. 'Validation-Training Split' indicates the fraction of training data withheld for hyperparameter optimization. Hyperparameters are parameters whose values shape the learning of the model. It is always best practice to determine the best combination of hyperparameters on a dataset that is separate from the testing data to mitigate overfitting.

b. Second, we press the "validate dataset" button.

Note: This step ensures that data is appropriately structured as described in the preceding sections labeled critical. Specifically, this step ensures the data is binary (contains only 0's and 1's), that every neuron fires at least twice in the testing and training datasets, and that each observation contains at least one neuron firing. In the event of failed validation, the validator will attempt automatic solutions if applicable. Successful validation will be confirmed to the user in the logging console (Figure 2B). Otherwise, an error will appear.

Note: Pressing the 'Run Model' button will launch the entire analysis with a single-click (including validation). However, in this tutorial, we will proceed manually through each step.

Structural learning

Timing: 1–5 min

The objective of the structural learning step is to generate a set of structures which approximate the functional connectivity of the neural circuit. We accomplish this by conducting neighborhood-based regressions implemented through highly-efficient elastic nets. The most optimal structure for the graphical structure will be determined during the hyperparameter optimization step.

6. Click on the 'Training' tab to proceed to the training stage.
7. Click on the "Structural Learning" tab to open the structural learning submenu (Figure 3A).
8. Setting Structural Learning Parameters.
   a. 'Parallel Learning' indicates whether to learn the set of structures in parallel. This is step can save considerable time at the expense of using more memory. By default, it is set to 0 (off).
   b. 'Alpha' indicates the balance of ℓ_1-to-ℓ_2 regularization in the elastic net during structural learning. We set this parameter to the default value of 1. This parameter does not need to be changed but is provided for advanced users.
   c. 'Number of sLambda' indicates the number of distinct sLambda values used to minimize the structural learning objective.
   d. 'Minimum sLambda' and 'Maximum sLambda' set the range from which sLambda parameters are selected. By default, this range is set to 1e-05 to 5e-01. These parameters should rarely need changed.
   e. 'sLambda distribution' sets the distribution used to randomly select sLamba values. By default, this parameter is set to "1" to utilize a log distribution. This allows for more samples to be selected at smaller sLambda where the complexity of output structures is greater. Users should only set this to parameter to "0" (uniform distribution) if structural learning is prohibitively time-consuming for a particular dataset. Setting this parameter to "0" will improve performance at the expense of accuracy.
   f. 'Edge Constraints' controls the connectivity of user-defined feature nodes within the learned structure. By default, this parameter is set to "1" to constrain the structure such that user-defined feature nodes cannot connect to each other. This parameter does not need to be changed but is provided for advanced users who may be interested in exploring interactions between user-defined features.

CellPress
OPEN ACCESS

Figure 3. Structural learning and parameter estimation

(A) Structural learning panel with an interface for initiating structural learning and a drop-down box highlighting a pseudo-random selection of s-lambda values to be passed to parameter estimation.

(B) Parameter estimation panel with an interface for initiating parameter estimation and a drop-down box highlighting selected p-lambda values.

(C) Console feedback during parameter estimation reporting progress and computation time.

g. 'Absolute Value Ranking' formulates structures based on the absolute value of their coefficients rather than their positive magnitude. We will set this parameter to 0. However, it could be useful in mixed datasets including excitatory and inhibitory neurons and has been left open to the user.
h. 'Number of Seed Structures' determines how many structures to randomly select as seeds for parameter estimation and hyperparameter optimization.
i. 'Density' sets an optional threshold for the density of the learned structure. The number of edges kept in the structure is equal to this value times the number of possible non-zero edges.

9. Press the run Structural Learning to generate a set of structures to pass to parameter estimation and hyperparameter optimization.

Note: A progress bar indicating progress will be displayed in the terminal.

Parameter estimation

Timing: 10–30 min

The object of this step is to infer the potentials which dictate the magnitude of the interactions identified during structural learning. This objective is accomplished through maximum likelihood estimation.

10. Click on the "Parameter Estimation" tab to open the parameter estimation submenu (Figure 3B).
11. Setting Parameter Estimation parameters.
    a. 'Implementation Mode' sets the implementation of parameter estimation. Changing the default "1" to "2" conducts parameter estimation on multiple potential models in parallel at the expense of increased memory requirements. Setting the mode to "3" or "4" conducts sequential or parallel parameter estimation on seed models only. This setting is useful to get a quick gauge of the amount of time it takes to learn models of a particular size or composition before launching a more-expansive optimization.
    b. 'fVal Epsilon' sets the convergence criterion and is best left at default. It is provided to allow advanced users to relax the criterion on exceptionally large or complex datasets.
    c. 'Reweight Denominator' sets the method used to relax the approximation of the partition function to facilitate learning. It is best left at the default. It is provided to allow advanced users to tailor performance when using exceptionally large or complex datasets.
    d. 'Number of pLambda' sets the number of pLambda values used as seeds during parameter estimation.
    e. 'Minimum pLambda' and 'Maximum pLambda' set the range from which pLambda values are selected.
    f. 'pLambda Distribution' sets the distribution from which seed pLambda values are selected. Set this parameter as "0" for a uniform distribution or "1" for a log space.
    g. 'Max Iterations' sets the upper limit for the number of iteration to allow during parameter estimation. It is best set at default, but can be decreased to improve performance for large or complex datasets
    h. 'Max Time' sets the upper limit for the amount of time spent on a single model during parameter estimation. While the example dataset will converge relatively quickly, I can be advantageous to set this parameter to a few thousand seconds to ensure poorly-suited models don't' absorb valuable computation time.
    i. 'Print Interval' sets the regularity of user feedback during learning. It does not affect performance and can be left at the default.
    j. 'SMBO Max Evaluations' sets the maximum number of models to evaluate during hyperparameter estimation. The default value is usually sufficient.
    k. 'SMBO Max Time' sets the maximum amount of time permitted to optimize the hyperparameters.
12. Press the "perform parameter estimation" button to estimate parameters.

Note: Verbose feedback will be printed in the command window (Figure 3C).

Model evaluation

Timing: <5 min

The objective of this step is to identify the best model from the pool of all learned models. The best model is selected by assessing the performance of each model on a withheld dataset.

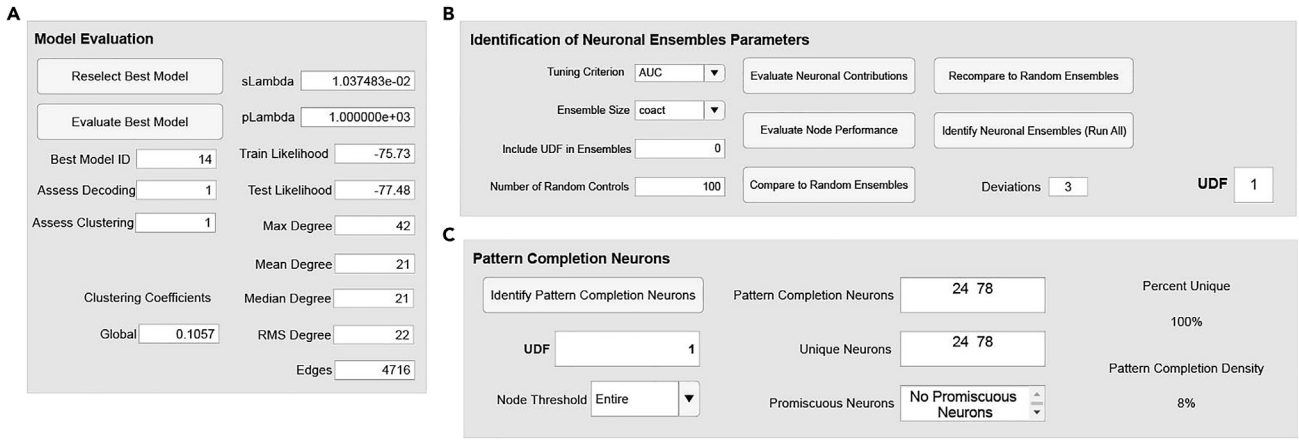


Figure 4. Leveraging learned models for identifying pattern completion neurons

(A) Model evaluation panel displaying the statistics and properties of the optimal model. Note the button to evaluate the model's ability to predict the occurrence of user-defined features.

(B) Panel for identifying neuronal ensembles.

(C) Panel for identifying pattern completion neurons.

13. Click on the "Evaluate Model" Tab to open the model evaluation submenu (Figure 4A).

Note: The best model is automatically selected following parameter estimation.

14. Press the "Evaluate Best Model" button to evaluate the learned model.

Note: A table will be produced describing the performance of the model in predicting user-defined features. An interactive visualization will be produced allowing users to highlight specific nodes within the graphical structure to observe their connectivity.

Note: Select which UDF to visualize using the UDF edit field.

Identification of neuronal ensembles

Timing: <5 min

The objective of this step is to identify the groups of neurons (neuron ensembles) specifically relevant to each UDF.

15. Click on the "Identify Ensembles" tab to open the Identifying Neuronal Ensembles submenu (Figure 4B).
16. Setting identification of neuronal ensembles parameters.
    a. 'Tuning Criterion' sets the criterion for assessing the performance of individual neuronal nodes. The default criterion is 'AUC' for the area-under-the-curve of the receiver operating characteristic (true positive rate vs. false positive rate). In some cases, the dataset may be heavily imbalanced. That is, the empirical probability of a user-defined feature being present is low (~10%). In this case, the criterion ought to be set to 'PR' to utilize a precision-recall curve.
    b. 'Ensemble size' controls the size of random ensembles used for comparison. By default, the random ensemble size is equal to the size of the node with the highest degree. However, it can also be set to the size of the maximum coactivation of neurons or the size of the maximum coactivation of neurons during UDFs.

c. 'Number of Random Controls' sets the number of random ensembles used for comparison. The default value of 100 is sufficient, but the user may increase this value if desired.

17. Press 'Evaluate Neuronal Contributions' to quantify the individual contributions of individual neurons in the model.

Note: Here, the ratio of the log-likelihood of a model with and without the neuron active are compared.

18. Press 'Evaluate Node Performance' to predict the state of UDFs using the change in the contributions of individual neurons.
19. Press 'Compare to Random Ensembles' to compare these predictions to the performance of random ensembles and identify neuronal ensembles.

Note: Neurons are added to an ensemble when they surpass a threshold calculated using the performance of randomized ensembles.

Note: The threshold for ensembles neurons can be modified and new ensembles extracted through the 'Recompare to Random Ensembles' buttons.

Identifying pattern completion neurons

Timing: <5 min

The objective of this step is to identify the pattern completion neurons for each UDF.

20. First press the 'Identify Pattern Completion' tab to open the Pattern Completion Neurons submenu.
21. To identify pattern completion neurons within the identified ensembles (if present), press the 'Identify Pattern Completion Neurons' button (Figure 4C).

Note: Pattern completion neurons are defined as those neurons surpassing a threshold for both predicting the UDF and the strength of their influence within the network—defined as the node strength, the sum of their Phi-11 coactivity potentials.

Note: The results will be automatically plotted. To select which UDF is currently plotted, change the 'UDF' field.

22. Press the export button to save and export results.

⚠ CRITICAL: The identity of ensemble and pattern completion neurons will be saved in cell arrays named 'ensemble_nodes' and 'pattern_completion_nodes'. The length of each cell array is equal to the number of user defined features. Each cell contains an index of the identified ensemble or pattern completion nodes. A snap-shot of all the calculated data, including sub-optimal models, is also exported. Descriptions of these exported variables and parameters can be found here.

EXPECTED OUTCOMES

PatMap is a simple graphical user interface that intuitively models neural circuit dynamics as probabilistic graphical models—where nodes represent neurons, and edges their functional connections (Figures 5A and 5B). PatMap can leverage the probabilistic dependencies within these models to identify the computational building blocks that underlie behavior (e.g., neuronal ensembles, Figures 5C and 5D). Unlike traditional dimensionality reduction or encoding methods, these

neuronal ensembles are described within the statistical scope of individual neurons. This biologically-meaningful modeling lends itself naturally to studies involving targeted manipulations, allowing informed manipulation of behavioral, circuit and ensemble activity. To this end, in its present functionality PatMap identifies pattern completion neurons within neuronal ensembles—these neurons have been found capable of activating complete neuronal ensembles through their robust influence on population activity (Figures 5E and 5F; Figure 6; Methods video S1).

LIMITATIONS

PatMap is currently a supervised method, and thus cannot detect encoding of unexpected stimulus features. Second, PatMap requires binary data. It follows that information may be lost when using fluorescence-derived data (e.g., calcium imaging) as it cannot incorporate magnitude. Also, our analysis is currently limited to detect zero-lag neuronal interactions, although we are exploring future versions that incorporate detecting temporal interactions in neuronal firing. Finally, while PatMap's computational and structured prediction performance is optimal, in practice it can become computationally expensive as model complexity increases.

TROUBLESHOOTING

Problem 1

PatMap returns repeated warnings regarding failure to converge during structural learning (Step 10).

Potential solution

This problem typically arises when the sequence of sLambda parameters is not well-suited. This problem can usually be solved by using a large sLambda minimum (i.e., using a smaller range).

Problem 2

PatMap is able to decode UDFs with good performance, but is unable to identify high performing ensembles (Step 19).

Potential solution

Some neural circuits may not utilize neuronal ensembles. In some case, neural circuits may dynamically encode the UDF in stochastic network activity that may be incorporated into the model. Nevertheless, this problem can also arise due to a high baseline of performance in randomized ensembles (often due to random ensembles that are very large). This problem is easily identified by observing the distribution of node performance on the 'Identifying Neuronal Ensembles tab' and solved by using a larger deviation during ensemble identification.

Problem 3

PatMap indicates my dataset is invalid but every sample and neuron has multiple spikes (Step 5).

Potential solution

If some neurons spike very infrequently, on occasion these spikes will all be grouped in the test dataset upon shuffling. This can be solved by removing these neurons or reshuffling the dataset.

Problem 4

I received a segmentation error and crash to desktop during parameter estimation and/or hyperparameter estimation (Step 12).

Potential solution

Select functions that require high-performance are C++. To make it easy for users, we have precompiled 'mex' functions that interface this performant code with MATLAB. Nevertheless, it is possible that there are particular combinations of Matlab and operating system releases for which these compiled functions aren't well-suited. In this case, we have prepared a recipe to easily compile these

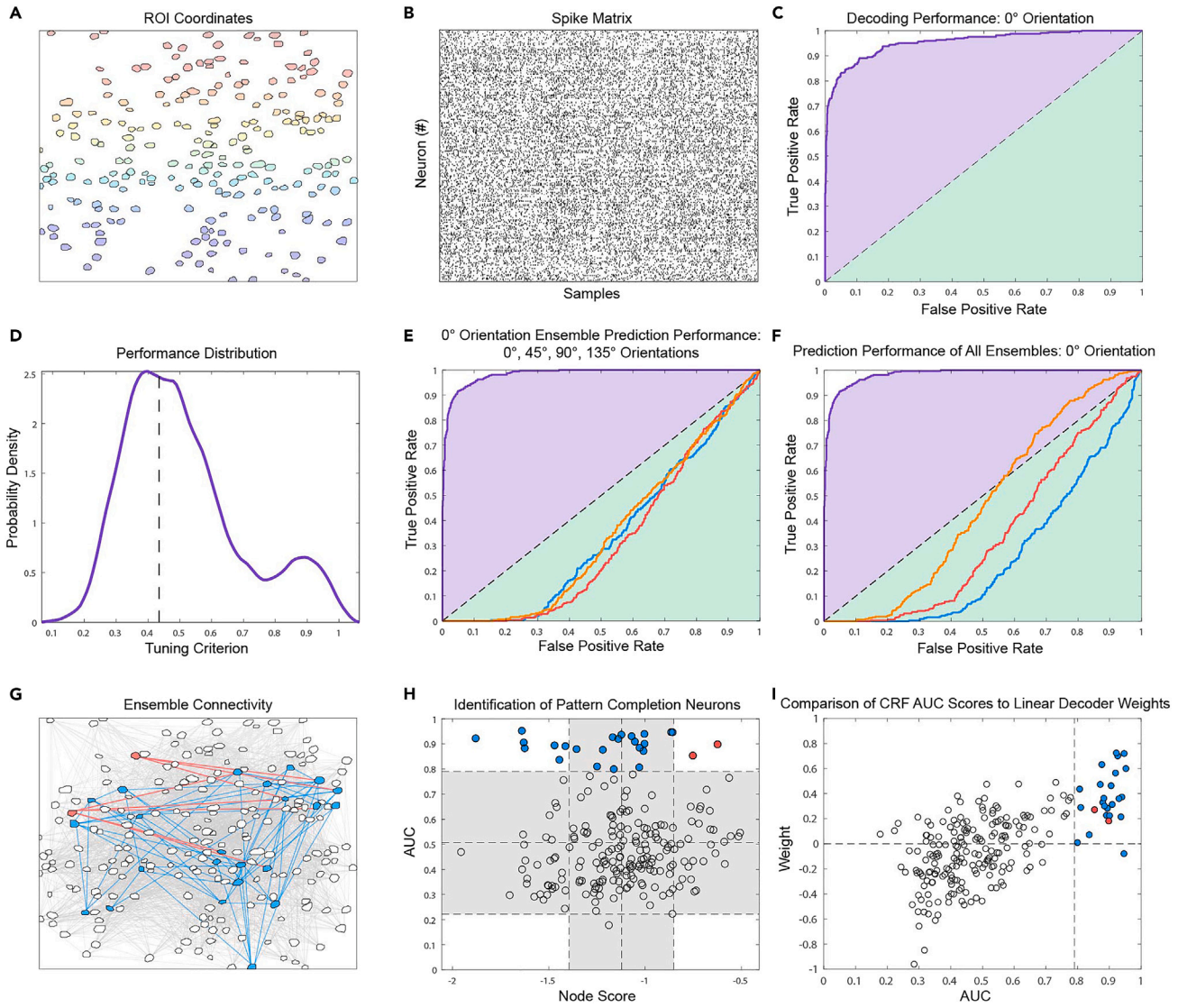


Figure 5. PatMap analysis of excitatory neurons in mouse visual cortex

(A) Spatial locations of neurons identified in example dataset.

(B) Example data spike matrix associated with identified neurons.

(C) Decoding performance of best model (16 models compared). The area-under-the-curve (AUC) of the receiver operating characteristic is 0.95. Dotted line represents the performance of a no-skill classifier.

(D) Distribution of tuning criterions for all nodes. Here, the tuning criterion was the AUC of a decoder trained to identify the presentation of a 0°-oriented drifting grating using the log-likelihood ratio of the model containing or not containing a specific neuronal node.

(E) Performance of 0° orientation ensemble in identifying presentation of 0°, 45°, 90°, and 135°-oriented drifting gratings. Purple (0°, AUC 0.98), red (45°, AUC 0.32), blue (90°, AUC 0.34), orange (135°, AUC 0.35). Dotted line represents the performance of a no-skill classifier.

(F) Performance of 0°, 45°, 90°, and 135°-orientation ensembles in identifying the presentation of a 0°-oriented drifting grating. Purple (0°, AUC 0.98), red (45°, AUC 0.34), blue (90°, AUC 0.26), orange (135°, AUC 0.47). The AUC of the 0° was 0.98. Dotted line represents the performance of a no-skill classifier.

(G) Spatial location and connectivity of 0° ensemble neurons (blue) and its pattern completion neurons (red).

(H) Pattern completion neurons are readily identified by plotting the AUC for a specific UDF and the node scores. The y-axis bounds represent +/- one standard deviation within the mean of random ensemble performance. The x-axis bounds represent 95% confidence intervals of the ensemble's node scores. The three pattern completion exceeds both these bounds.

(I) Comparison of identified ensemble and pattern completion neurons to the coefficient estimates or weights of a support vector machine trained to classify the frames in which a 0° oriented drifting grating was presented. Note that the identity of pattern completion neurons cannot be discerned by their utility in classification.

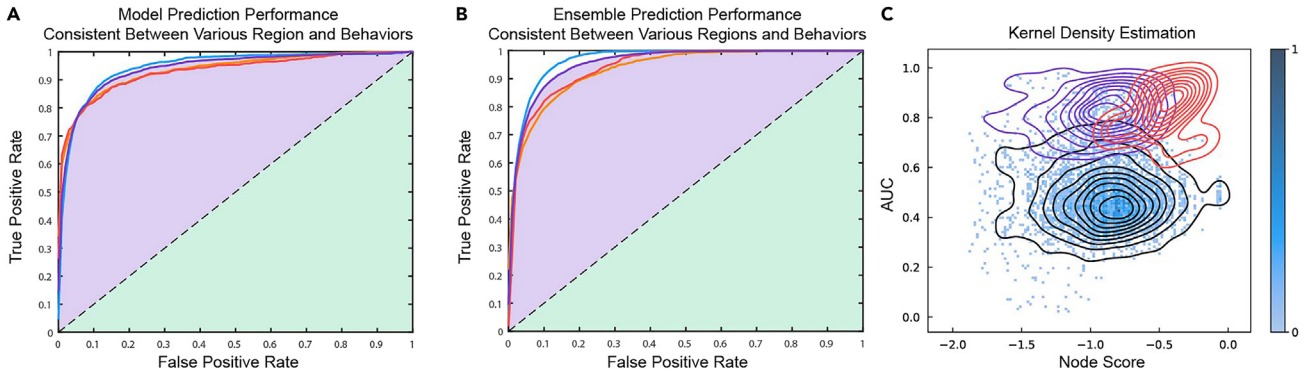


Figure 6. Broader applicability

(A) Decoding performance of best models trained to decode the orientation of drifting gratings in the visual cortex, the context during fear conditioning in the hippocampus, and decoding position in the hippocampus. Purple (mean, AUC 0.95), red (context, AUC 0.90), blue (position, AUC 0.95), orange (orientation, AUC 0.94), red: context, blue: position, orange: orientation. Dotted line represents the performance of a no-skill classifier.

(B) Decoding performance of identified ensembles trained to decode the orientation of drifting gratings in the visual cortex, the context during fear conditioning in the hippocampus, and decoding position in the hippocampus. Purple (mean, AUC 0.96), red (context, AUC 0.94), blue (position, 0.97), orange (orientation, AUC 0.93). Dotted line represents the performance of a no-skill classifier.

(C) Kernel density estimate of all neurons' AUC and node scores. Red: pattern completion neurons, purple: ensemble neurons, black: non-ensemble neurons, colorbar: density.

functions on your own. Simply open the 'Makefile' in the repository with a text editor, enter the path to your Matlab 'mex' file, open a terminal in the repository, and enter 'make'.

Problem 5

My dataset generates out-of-memory errors. How do I analyze my dataset (Step 12)?

Potential solution

The simplest approach reducing memory requirements is to lower the density parameter. This will reduce the complexity of learned structures by reducing the number of edges. Even small reductions in the density parameter (e.g., subtracting 0.01) can greatly reduce memory requirements in large datasets. A second approach is to reduce the number of neurons included within the model through random sampling or the utilization of feature selection methods. While reducing the number of samples is an option, the number of nodes is the primary determinant of the required amount of memory. Model structure is generally robust to the addition and subtract of individual nodes (see here).

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Darik A. O'Neil (d.oneil@columbia.edu).

Materials availability

This study did not generate new unique reagents.

Data and code availability

All datasets and code used during this study are available at www.github.com/darikoneil/PatMap. The accession number for this material is also available at Zenodo: 8145204.

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.xpro.2023.102543.

ACKNOWLEDGMENTS

This work was supported by grants from the National Institute of Mental Health (R01MH115900), National Science Foundation (2203119), and Vannevar Bush Faculty Fellowship (ONR N000142012828) to R.Y.; National Eye Institute (F32EY029161 and K99EY033974) to A.A.; and National Institute of Neurological Disorders and Stroke (F99NS134209) and National Science Foundation Graduate Research Fellowship to D.A.O.

AUTHOR CONTRIBUTIONS

D.A.O. wrote the manuscript, graphical user interface, and code. A.A. performed experiments, analyzed data, and edited the manuscript. R.Y. assembled the team; edited the manuscript; provided equipment, resources and funding; and directed and supervised the project.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. Carrillo-Reid, L., Han, S., O'Neil, D., Taralova, E., Jebara, T., and Yuste, R. (2021). Identification of Pattern Completion Neurons in Neuronal Ensembles Using Probabilistic Graphical Models. J. Neurosci. *41*, 8577–8588. https://doi.org/10.1523/jneurosci.0051-21.2021.

2. Yuste, R. (2015). From the neuron doctrine to neural networks. Nat. Rev. Neurosci. *16*, 487–497. https://doi.org/10.1038/nrn3962.

3. Buzsáki, G. (2004). Large-scale recording of neuronal ensembles. Nat. Neurosci. *7*, 446–451. https://doi.org/10.1038/nn1233.

4. Carrillo-Reid, L., Han, S., Yang, W., Akrouh, A., and Yuste, R. (2019). Controlling Visually Guided Behavior by Holographic Recalling of Cortical Ensembles. Cell *178*, 447–457.e5. https://doi.org/10.1016/j.cell.2019.05.045.

5. Carrillo-Reid, L., Yang, W., Bando, Y., Peterka, D.S., and Yuste, R. (2016). Imprinting and recalling cortical ensembles. Science *353*, 691–694. https://doi.org/10.1126/science.aaf7560.

6. Sutton, C., and McCallum, A. (2012). An Introduction to Conditional Random Fields. FNT. in Machine Learning *4*, 267–373. https://doi.org/10.1561/2200000013.

7. Ravikumar, P., Wainwright, M.J., and Lafferty, J.D. (2010). High-dimensional Ising model selection using ℓ_1-regularized logistic regression. Ann. Stat. *38*, 1287–1319. 1233.

8. Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. J. Stat. Softw. *33*, 1–22. https://doi.org/10.18637/jss.v033.i01.

9. Tang, K., Ruozzi, N., Belanger, D., and Jebara, T. (2016). Bethe Learning of Graphical Models via MAP Decoding. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, G. Arthur and C.R. Christian, eds. (PMLR).

10. Rother, C., Kolmogorov, V., Lempitsky, V., and Szummer, M. (2007). Optimizing Binary MRFs via Extended Roof Duality (IEEE), pp. 1–8.

11. Snoek, J., Larochelle, H., and Adams, R.P. (2012). Practical bayesian optimization of machine learning algorithms. Adv. Neural Inf. Process. Syst. *25*, 1–9.

12. Krishnan, S., Heer, C., Cherian, C., and Sheffield, M.E.J. (2022). Reward expectation extinction restructures and degrades CA1 spatial maps through loss of a dopaminergic reward proximity signal. Nat Commun *13*, 6662. https://doi.org/10.1038/s41467-022-34465-5.