READ ME

This a **beta version** of the CRF modeling approach developed in the laboratory of Rafael Yuste at Columbia University in coordination with the research group of Tony Jebara.

The program approximates constructs a probabilistic graphical model (conditional random field) that approximates the underlying neural circuit that generate a spike matrix (i.e., functional connectivity). This model can be leveraged to select pattern completing neurons, identify ensembles, and much more. This model is learned in three steps:

1.) **Structural Learning**—(the connectivity), which consists of an ultra-fast elastic net implementation (GLMNet) where alpha is 1 (i.e., L1-regularization).

2.) **Parameter Estimation**—(the strength of these connections), where we invoke the MLE-Struct approach developed by the Jebara research group.

3.) **Model Selection / Cross-Validation**—where we maximize the summed log-likelihoods of training and withheld datasets.

A NOTE ON MODULARITY:

Our implementation is designed to offer significant modularity, allowing for seamless integration of alternative methods, implementations, and exploratory analyses using partial-runs. Our current implementation utilizes the terminal of Linux-based OS as the master interface, though it would not be difficult to utilize Python instead (we do not use Python simply due to the difficulty of integrating Python and MATLAB on some systems). **All modeling functions are run in MATLAB or in C++ or FORTRAN *through* MATLAB.**

A NOTE ON PARALLELIZATION:

All structural learning is performed using a **single-core** to optimize performance, though parameter estimation can be distributed across **multiple cores** on a single machine or high-performance computing cluster (HPC). Please note that while HPCs by nature have many thousands of cores, the single-core performance for *many* HPCs is generally poor by modern standards. In our analysis computing, a single core on our standard analysis PC (i9-990K; 5.2 GHz) processes over 250% faster than a single core of our HPC. Depending on the number of models you are considering, and your internet speed, it may be faster to parallelize your modeling on a single machine. Likewise, the blocks of the Franke-Wolfe optimization within our implementation are not distributed. The parameter estimation of our implementation has so far been extremely fast (typically in a few seconds), and therefore invoking a parallel-distribution is rather trivial. Nevertheless, if you intend to use extremely-large datasets (e.g., 15,000 neurons) you may

find using such a parallelization fruitful. Likewise, some secondary analysis can be implemented using GPU-parallelization, though in our datasets the swap to the GPU takes longer than the actual processing time.

USER-DEFINED FEATURES:

User-defined features can be incorporated, provided they are appropriately binarized. Doing so many be advantageous for secondary analysis, for structured-prediction, decoding, and ensemble-identification purposes.

REQUIREMENTS:

MATLAB 2019 or greater (must have symbolic link; typically does)

Signal Processing Toolbox

Linux-based OS (Ubuntu LTS recommended; Windows forthcoming)

THIRD-PARTY DEPENDENCIES:

QPBO 1.32

GLMNet

INSTALLATION:

Execute beta_installer script in the terminal

That is, type *bash beta_installer.sh* in the terminal

DEMO SCRIPT:

To run demo script, Execute Single_Core script at *default parameters* using the data *darik_demo* dataset

GLOSSARY

VARIABLES: (incomplete)

data	Binary spike matrix in the form of frames by neurons (best constrained to high-activity frames or binned)
UDF	User-defined features in the form of frames by features (e.g., a visual stimulus)
coords	Coordinates of neurons in the forms of neurons by coordinates (2D or 3D; optional)
edge_potentials	Symmetric matrix of edge potentials
node_potentials	Column vector of node potentials
F	2xNode Count matrix with node weights (1 0)
G	4xEdge Count matrix with edge weights (00 01 10 11)

core_crf	High ranking nodes
PCNs	Neurons selected to be core pattern completers using PAPS score
s_lambda	Regularization parameter for structural learning
p_lambda	Regularization parameter for parameter estimation