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.

This implementation only accepts data in a special format. **Binary** spike data must be in the form $i = time_i, j = neuron_j$. User-defined features are optional, and likewise must be binary and in the format $i = time_i, j = UDF_j$. Neuronal coordinates of the form $i = neuron_i, j = coordinates\ (x,y \text{ or } x,y,z)$, can be included for plotting or mapping to the imaging space for future stimulation. These variables must be saved in a .mat file with a dedicated filename (e.g., 'darik_demo.mat')

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

MAJOR VARIABLES:

data	Binary spike matrix in the form $i = time_i$, $j = neuron_j$ (best constrained to high-activity frames or binned)

UDF	User-defined features in the form *$i = time_i$, $j = UDF_j$.* (e.g., a visual stimulus ON/OFF; optional)
coords	Coordinates of neurons in the forms *$i = neuron_i$, $j =$ coordinates (x,y or x,y,z; optional)*
filename	Filename containing data (and optionally UDF / coords) in the .mat format (e.g., 'darik_demo.mat')
theta	Learned parameters of individual model
edge_potentials	Symmetric matrix of edge potentials
node_potentials	Column vector of node potentials
F	2xNode Count matrix with node weights (ϕ1 ϕ0) & *$j = node_j$*
G	4xEdge Count matrix with edge weights (ϕ00 ϕ01 ϕ10 ϕ11) & *$j = edge_j$*
logZ	Log-partition function
max_degree	Maximum number of edges for single node
median_degree	Median number of edges per node
mean_degree	Average number of edges per node
rms_degree	Root mean square edges per node
reweight	Reweight value
train_likelihood	Training log-likelihood
test_likelihood	Testing log-likelihood
results	File of modeling results (best_model only)
auc	Area under the curve of the receiver operating characteristic, *$i = node_i$ $j=UDF_j$*
auc_ens	AUC for randomized control ensembles
core_crf	Cell-array of core neurons for each user-defined feature
Epsum	Sum of ϕ11 for each node
PCNs	Cell-array of neurons selected to be core pattern completing neurons using PAPS score for each user-defined feature
PAPS_INDEXED	Index of PAPS score for each neuron in each UDF
params	Structure of modeling parameters
best_model_s_lambda	Regularization parameter for structural learning in best model
s_lambda_range	Number of λ_s to generate structures for (**minimum 100**)
s_lambda_min	Smallest λ_s
s_lambda_max	Largest λ_s
logspace	Distribution of λ_s within designated range (0 = linear space, 1 = logspace)
s_lambda_sequence_LASSO	Sequence of λ_s fed into GLMNet
LASSO_options	Options fed into GLMNet
num_structures	Number of λ_s to stochastically select for parameter estimation **note that the smallest and largest λ_s outside the warm-start are always selected to ensure proper bracketing*
s_lambda_sequence	Sequence of λ_s passed to parameter estimation
learned_structure	Cell-array of learned structures
GLM_array	Cell-array of GLMNet objects for each neighborhood
variable_groups	Cell-array of neighborhood compositions
hyperedge	2 = constrain structures so there are no UDF-UDF edges

structures	Collection of learned structures
best_model_p_lambda	Regularization parameter for parameter estimation in best model
p_lambda_count	Number of λ_p to assess within designated range
p_lambda_min	Smallest λ_p
p_lambda_max	Largest λ_p
p_lambda_sequence	Sequence of λ_p passed to parameter estimation
Num_Nodes	Number of neuronal nodes
UDF_Count	Number of UDF nodes
split	Percent of data included in training dataset
x_train	Training dataset of form $i=time_i$ $j = node_j$
x_test	Testing dataset of form i=timei j = nodej
compute_true_logZ	1 = use JTA to compute logZ (not recommended)
models	models for parameter estimation
model_collection	Object containing collection of models and relevant parameters; interfaces with BCFW object for parameter estimation
model_parameters	Collection of parameters for this modeling run
reweight_denominator	Variable used as denominator in reweighting (mean degree, median degree, max degree, rms degree)
BCFW_max_iterations	Maximum iterations for individual model's parameter estimation (BCFW)
BCFW_fval_epsilon	Epsilon value for parameter estimation (BCFW)
printInterval	How many intervals before printing (BCFW)
printTest	How many intervals before testing (BCFW)
MaxTime	Maximum iteration time for individual model's parameter estimation (BCFW)
name	Filename
Filename	Filename with extension
data_directory	Directory of data
source_directory	Directory of repo
exptdir	Directory of modeling run for this experiment

MAJOR FUNCTIONS & SCRIPTS: