admin@chalearn.org Edit this form

31 responses

View all responses Publish analytics

Summary

Team name

AAAGV
gaucho81
Ildefons Magrans
Test
Algoasaurus
DJMN
Sandro
Gideon & Alex
Inicalo
Technology Trends Observation Team
scharfy
James Tee
Matthias Ossadnik
Selfish Gene
MortalKolle
Alexander N & vopern
:-)
dhanson
Rangel Dokov
Lejlot & Rafal
Piotr Kuchta
Nitai Dean
killertom
Gábor S
kazAnova
Konnectomics
kid b
SM
Lukasz 8000
ULB MLG
Alex Stanciu

Team website URL

http://chalearn.org
http://ccsb.fudan.edu.cn//
mlg.ulb.ac.be
http://www.kazanovaforanalytics.com/
https://github.com/asutera/kaggle-connectomics
https://www.linkedin.com/in/bcragin

Team Leader Name

Yasunobu Igarashi
Marios Michailidis
Lukasz Romaszko
Alistair Muldal
Gianluca Bontempi
Ildefons Magrans de Abril
James Tee
Isabelle
Damir Jajetic
Matthias Ossadnik
George Mohler
Bruce L Cragin
Gideon Rosenthal
Antonio SUTERA
Nitai Dean
Rafal Jozefowicz
Chenyang Tao
Sandro Vega Pons
Alex Stanciu

Contact email

gbonte@ulb.ac.be
kazanovassoftware@gmail.com
yasunobu.igarashi@gmail.com
gidonro@gmail.com
alistair.muldal@pharm.ox.ac.uk
scharf.timothy@gmail.com
bcragin@capcityresearch.com
james.tee@gmail.com
nitaidean@gmail.com
djajetic@gmail.com
lukasz.romaszko@gmail.com
alexcstanciu@gmail.com
a.sutera@ulg.ac.be

shafaq12@gmail.com
cytao@fudan.edu.cn
ildefons.magrans@gmail.com
A.Niederbuehl@gmail.com
georgemohler@gmail.com
sv.pons@gmail.com
asada@asdasd.asdsa
ossadnik.matthias@gmail.com
rafjoz@gmail.com

Team Leader Address and phone number

Room 2303, East Wing Guanghua Tower, 220 Handan Road, Yangpu, Shanghai +86 - 189 1701 3615
Timothy Scharf 312-771-9002
Department of Pharmacology University of Oxford Mansfield Road Oxford OX1 3QT United Kingdom Tel: 01865 281113
Nitai Dean 347 Transylvania Park Lexington , KY 40508
Address: Tel Aviv Israel Phone: 972506818336
Viale Verona 49, Trento, Italy 38123 +393427172901
Matthias Ossadnik Elbestr. 15 60329 Frankfurt am Main Germany
Math/CS Dept Santa Clara Unversity 805-252-0656
650 W 42nd Street, Apartment 3404 10036 New York, New York Phone number: +1 347 977 2013
Lukasz Romaszko Uilenstede 102F 750 1183AM, Amstelveen, The Netherlands +31616028720
asdasd
GSM: +32 494 48 37 78 Office: +32 4 366 27 18 401, Rue de l'arbre sainte barbe 4000, Liège Belgium
1357 US Rte 10 Lempster, NH 03605-3303 Tel: Mobile: 1-603-757-3480 Home: 1-603-863-3848
Home Address: Van Bortonnestraat 56 1090 Brussels Belgium Phone number: +32 487 41 4833

Other team members

Linus Schumacher Peter Humphreys Alvaro Tejero-Cantero Sam Harrison Timothy Lillicrap
NA
Daniel Abramson
asds
Wojciech Czarnecki
Maja Novak
Maxime Flauder <maxime.flauder@student.ecp.fr>
Arnaud JOLY (+32 498 61 00 01) Aaron Zixiao QIU (+32 494 48 48 00) Gilles LOUPPE (+32 473 59 59 68) Vincent FRANCOIS-LAVET (+32 494 59 17 17)
Alexander Lebedev

Method summary

Title of your contribution

Chi SQ + discretized correlation
ICA-based connectivity
Simple Correlation-based Model
DJMN
GTE with deblurring and fast non-negative deconvolution
-
Regression on network deconvolution based features
asd
An SVM approach to the Connectomics Challenge
Nitai Dean Method
Pearson Correlation with discretization
Deep CNN for Time Series Correlation Measurement
Neuronal Connectivity Reconstruction from Calcium Imaging Signal using Random Forest with Topological Features
Csiszár's Transfer Entropy and Integration of Connection Matrices
Third Place Solution of the Connectomics Challenge
hitmat
scores_test
Optimized Logistic Regression for Network Connection Detection
shafaq.
feature engineering is extremely important
KazAnova Contribution
direct connections
Connectomics - TEAM AAAGV
Quick and Dirty
Oopsi thresholded network deconvolution
Pearson correlation coefficient
Random forest using GTE, IGCI, and Granger
Ensemble method for inferring neural network connectivity using time lagged inverse cross variance features
mIMR
A really simple correlation-based method
ConnetomicsSubmit

General description

My method uses Pearson's correlation between pairs of neurons, but instead of submitting raw correlation scores for each X -> Y pair, for each X neuron I gave a score of 1 to the X -> Y connection where Y is the neuron with which X has the highest correlation, a score of 0.999 to X -> Y where Y is the neuron with the second highest correlation with X, and so on. This effectively "normalizes" the submitted values between neurons, and it works well because there are no huge differences in the number of connections the neurons have. This simple "hack" took my public leaderboard score from 0.87337 to 0.90001. The rest of the improvement is attributable to filtering out periods of high network activity, and also using the above mentioned correlation ranking for the reverse direction, Y -> X.

The method is chiefly based on logistic regression on binarized neuron activity time series. Logistic regression is a standard approach for estimating directed or undirected graphical models (see e.g. Hastie et al., "The elements of statistical learning"). The regression is run for each neuron, using the signals of the other neurons as predictors. Scores for the directed connections are derived from the regression weights of the predictors. This basic method is improved in two ways. First, in a preprocessing step, the fluorescence is binarized applying a threshold to the first (discrete) derivative of the signal. When the neuron is used as a predictor, this binary signal is transformed into a score that incorporates information from neighboring time steps as well. Parameters for the transformation were obtained by optimizing the logistic regression scores on the training data. In order to improve prediction performance, up to eight different parametrizations have been obtained by optimizing different random subsamples of the training data. Second, the logistic regression scores for the optimized parametrizations are augmented by additional available information such as neuron firing rates, and fed as features into variant of a gradient boosted decision tree classifier. The final score for each connection is obtained from this classifier. Since this classification problem is where unbalanced, the existing implementation within scikit-learn has been extended to use random undersampling similar to and inspired by the RUSBoost algorithm (Seiffert et al., ICPR 2008. 19th International Conference on Pattern Recognition, 2008.) that adapts the AdaBoost algorithm to problems with large class imbalances.

The method is based on a speculative derivation of pairs of neurons. The expected connectivity is calculated solely from the initial neuron being considered

Data was preprocessed using fast-Oopsi (1) to infer spike trains. Then, rows were thresholded for highest values and top 700 rows were chosen. Next, correlation matrix was calculated using these top rows. Finally, Network deconvolution (2) was computed to distinguish direct dependencies in networks. The combination of these methods and the fact that only 700(!) rows are enough to produce top ranking result is a novel and provide great insight on the nature of the data. This method is very computationally efficient, and could help to infer huge networks relatively easily. Thus, also it is not ranked at the very top of the leader board it may prove to be very useful in a real world settings, were a fast and accurate analysis is needed.

I've extracted many different simple pairwise features, such as cross correlation between filtered discretion signals (xcorr (h1*d1(t), h2*d2(t))) at different neural activety regimes, and other more holistic features such as PCA/kmeans type features, and sent them all to a RF regressor.

asd

My solution was basically computing a Pearson Correlation of discretized neuron activities. The best threshold for the discretization I found was 0. I used Python and scipy.stats.pearsonr to implement the above. Unfortunately, I didn't have time to come up with a more sophisticated solution.

The method is basically a combination of existing techniques. The main three steps are: - Preprocessing: - Derivative: We compute the discrete difference along the first axis. - Binarization: We threshold the resulting time series with a threshold t. - Feature extraction: - We compute the correlation matrix of all time-series belonging to a graph and normalize it in such a way that all correlation values for every node are scaled to [0, 1] - We apply the Network

Deconvolution (ND) algorithm on top of the correlation matrix. We wrote our own python implementation of this algorithm based on the original paper [1] and the publicly available matlab code [2]. This algorithm depends on a scaling parameter b. - For each graph in the training data (just normal-1, normal-2, normal-3, normal-4) we compute a set of features by computing a correlation matrix with the following threshold values t = {0.10, 0.12, 0.14, 0.16, 0.18} and for each of the 5 correlation matrices we apply the ND algorithm with 5 b values, b = {0.87, 0.90, 0.93, 0.96, 0.99}. By unfolding the 25 matrices and concatenating the 25 resulting vectors, we end up with 25 features for each edge in the graph, i.e. with a matrix of (samples, features) of size (1000000, 25). - Following the previous procedure we have four training sets (X1,y1) computed from normal-1 graph, …, (X4, y4) computed from normal-4 graph. The same procedure is applied on the two test sets obtaining (Xv, yv) from the valid data and (Xt, yt) from the test data. - Regression: - We train a Stochastic Gradient Descend Regressor (using the scikit-learn implementation sklearn.linear_model.SGDRegressor) on each training dataset. We apply the four regressors on the two test datasets (Xv, yv) and (Xt, yt). - From the previous step we get 4 prediction for the valid and test problems respectively (p1_v, p2_v, p3_v, p4_v) and (p1_t, p2_t, p3_t, p4_t). The final prediction is obtained by simply averaging the values in the four predictions i.e. pv = (p1_v + p2_v + p3_v + p4_v) /4 and pv = (p1_t + p2_t + p3_t + p4_t) /4. Finally both predictions are normalized to [0, 1] before submission. All the code was written in Python and it only depends on Numpy and Scikit-Learn. [1] http://www.nature.com/nbt/journal/v31/n8/full/nbt.2635.html [2] http://compbio.mit.edu/nd/index.html

Idea If we have distinct characteristics (possible predictors), calculated for each pair of neurons in a network, we can treat this as a set of observations in R^d. As we know the correct topologies for 4 networks similar to the test cases: normal-1, normal-2, normal-3 and normal-4, we can create 4,000,000 training pairs. We can now look at it as a binary classification problem and forget for a moment that the responses are not independent on each other. The classifier should: - work with a big dataset of 4,000,000 points - be able to see complex patterns in rather low dimensional (<200) space - work well with binary classification - be able to estimate confidence of the predicted value (not necessarily probability). We had tried a variety of different methods, including logistic regression, lasso, support vector machines, etc, but we have achieved the best results using random forests. It is a simple algorithm and yet able to extract relatively complex structures from the data. Due to its highly parallel nature, we could explore lots of different ideas for features in a limited timeframe of the competition. Testing methodology For evaluating features we use simple Leave-one-out (LOO) cross validation: for each k in {1,2,3,4} X = concatenate(X_i if i!=k) y = concatenate(y_i if i!=k) model = train(X, y) score[k] = auc_roc(model.confidence(X_k), y_k) return mean(score) and select the model with biggest such AUC ROC estimation. Features For any of the features used by the final algorithm, we used a very simple spike inference schema, i.e. we adopted the discretization method provided by the authors of the competition with different parameters (discretization thresholds and the number of bins). We also filtered out parts of the input in which lots of different neurons were spiking at the same time parameterized based on the mean activity across neurons (burst thresholds). This gave us the second level of parameterization for different algorithms. After the preprocessing we had time series with only 2 or 3 distinct values (depending on the binning method used). With that, we tried a bunch of base methods for the connection indication: - Correlation between the series - Mutual information with Gini function (IGG) - Mutual information with Entropy function (IGE) and a few others. The best of them, as a standalone feature, was able to achieve around 90% on the public leaderboard. A combination of them, parameterized with discretization and burst thresholds, was easily able to reach 91.5%. Up to this point, the model was not taking into account the fact that the samples are not independent on each other and thus was not able distinguish direct vs indirect connections. What gave us a big boost in performance were features created by transforming the existing predictions of individual methods that were exposing the topological properties of the problem. The simplest of those is the difference between the feature value and its transposition: given feature f[i, j] return f[i, j] - f[j, i]. It is able to extract whether the opposite edge is ranked higher by the method, which is important to know as most of the connections were “one-way”. Some other types of transformations that proved to be useful: - closure features - given feature f[i,j] we also computed max(sqrt(f[i,k]f[k,j])) which measured the potential other neuron k being responsible for "correlation" between ith and jth - relation based - i.e. f[i,j] / max(f[:,j]) measuring how strong is the relation in comparison with the strongest one - combination of the above, like f[i,j] / sum(f[i,k]f[k,j]) which more or less measures the ratio of the indicator strength as compared to the markov closure of length 1 (assuming f[i,j] is some kind of transition probability estimate). We tested dozen of such features and this, along with finding optimal random forests parameters, gave us a model that had competitive performance. In order to minimize the amount of required meta parameters fitting (like tree size, minimum number of elements in the leaf) we used the following scheme of adding new features to our representation: For new feature f Add f to representation R creating R’ Retrain model with R’ and evaluate LOO CV[R’] score If LOO CV[R’] < LOO CV[R] then reject f Else remove from R’ k worst features, where k=|R|-|R’|, and by worst we mean worst in the sense of the features’ importance of Random Forests This way our representation had a constant size and so optimal meta parameters were also rather constant. Remarks To the best of our knowledge using a binary classifier with topological features is a different approach than taken by other competitors and to what we could find in the existing literature. It has also some interesting properties: - Feature generation and training / prediction of the random forest algorithm is fully parallelizable and it scales linearly with the number of processors available (up to the point of training one tree, which is on the order of minutes). - Prediction for the edge between nodes i and j depends only on the metrics for neighbours of i and j. Due to its local nature it has potential to be applicable in working with much larger graphs since we are not required keep the whole graph in memory. We just need to be able to compute metrics for the neurons that are “close enough” to i and j. - It makes very little assumptions about the underlying neurobiological setting of the problem. In fact, the only part that’s related to neuron activities is initial filtering of the data and the discretization (which is very simple). The model behaves very well even though the spike inference is very basic and far from perfect. - It is easy to explain and to implement. - It doesn’t assume the graph to be undirected as graphical lasso algorithm does and thus can extract directed relationships between neurons. - It has low computational complexity. With our optimized implementation in python we can generate a base feature (like IGG, IGE, CORR, GTE, ..) in less than 30 minutes on one processor using less than 2GB of memory (most of the time). Because it’s using only a little amount of memory we were able to generate all the features used by the final algorithm within 48 hours on one machine (we need 6 cpus and ~15GB of RAM). The final model used more than 500 different generated features. This is orders of magnitude better than the original implementation of GTE and others in the TE-Causality package and the sample code in Matlab provided by the authors of the competition.

i only joined 2 days prior to the competition's end so I did not have much time to try different things . In the end I used: LibLinear's L2 logistic Regression (http://www.csie.ntu.edu.tw/~cjlin/liblinear/) In terms of featuresL 1)I used the the standard discretized correlation as per the benchmark 2) euclidean distances of the neurons' locations 3) multiple features derived by breaking down the full time series of 2 neurons in to 100-rows chunks and calculating average correlations and other basic descriptives,

First we generalize the Transfer Entropy (TE) to more generalized Csiszár's Transfer Entropy (CTE) by replacing the KL-divergence in the TE with more general Csiszár's f-divergence. Specifically, we used the Alpha-divergence (See Wikipedia's entry on f-divergence). By varying the parameter, we obtain a family of transfer entropy measures. We found that CTE give better performance compared to baseline TE with instant influence. To our knowledge, no one has generalize TE with Csiszár's f-divergence. CTE alone does not give optimal prediction. It's necessary to incorporate the information in the training set to achieve better discriminative power. We calculate Bayesian posterior and optimal linear combination through cross validation to improve the connection matrices obtained. Also, we incorporated some simple statistics such as linear correlation, delayed linear correlation calculated from time series in different dynamical regime to improve the resulting matrices obtained from CTE. Preprocessing of the Calcium Waves is also important. We use both the simple discretization and more sophisticated oopsi-package to preprocess the raw data. The parameters are optimized to give best performance in the training data. We note the importance of degree regularization of the network. Since the degree distribution follow certain rules, we can regularize the obtained connection matrices using the empirical distribution of in and out degree distribution.

GBRT

Compute the Pearson correlation coefficients for discretized data.

Nothing exciting. Two-bin discretization of the fluorescence data (after removal of high average fluorescence, presumably flaring-associated, time periods), followed by lag-zero correlation. A simple two-adjustable-parameter linear correction, in terms of the row and column averages of the correlation matrix, was then applied to each matrix element. The resulting adjusted value then served as the prediction of the "strength" of the associated connection. NOTE: I also tried other approaches, including ones based on Generalized Transfer Entropy; they just didn't make it into my final submission. Should I pursue additional work on this subject (e.g. a workshop presentation) it would probably deal with the comparison of correlation- and entropy-based methods, trying to identify circumstances when each one does better than the other. My answers on the following page of this Fact Sheet should be understood as applying to the full set of all models I tried in the competition rather than to the final selected model alone.

The core principle of our approach is to recover an undirected network by estimating partial correlations [1] for every pair of neurons. In particular, this approach is well-known for identifying first-order interactions (i.e., direct connections in the network) from higher-order interactions (i.e., indirect connections). In order to increase the performance, the raw data is preprocessed by i) filtering the time series and ii) re-weighting the samples to take into account the number (and the intensity) of burst peaks [2]. As a last step, and to obtain slightly better results, we asymmetrize our score matrix by trying to determine (heuristically) the causality.

We used an information theoretic filter selection method (mIMR) to rank for each neuron the most important ones.

I used the discretized correlation method mentioned as one of the benchmarks and a chi-sq test on the counts of coincident spikes/no spikes between pairs of time series. The two values were combined with a logistic regression.

random forest and linear models with some hand engineered features of discrete signals

Our method essentially consists of GTE with some slightly fancier pre-processing. We begin by trying to remove spurious correlations between nearby neurons that arise due to the optical blurring artifact. Given that noise is only added to the fluorescence traces prior to blurring, if we knew the blurring parameters, the amplitude (A) and length constant (lambda), then the blurring transformation is perfectly invertible. We set a threshold and only look at frames where the average fluorescence (across cells) is less than the bottom 5th percentile. In this regime the signal is dominated by the Gaussian noise which was added prior to blurring. When we examine the relationship between the Pearson correlation coefficients for each pair of cells and their distance from one another, we find that it is well-approximated by a Gaussian function. Since connection probabilities do not vary with distance, we know that this structure must arise due to the blurring. By fitting this function we estimate the A and lambda values, construct the mixing matrix, then solve for the unblurred signals. Next, since only spikes can be causal, we tried to obtain a better estimate of the true spike trains from the unblurred fluorescence. We wrote an optimised Python implementation of fast non-negative deconvolution (Vogelstein et al. 2010) for this purpose. Finally, we inferred connection weights using generalized transfer entropy, more or less as described in (Stetter et al 2012). We conditioned on the mean fluorescence level, and threw out time bins where this was greater than 25% of the maximum. We achieved our best results using a Markov order of 1 and allowing same-timebin interactions.

Material: GTE, IGCI, and Granger features Method: Randomforests

We used around 24 features, most of which were constructed using the regularized inverse cross variance matrix on spike times at different time lags and spike thresholds (all time series were deconvolved first to remove the simulated light scattering effects). We also included GTE (calculated using the provided sample code). We then used an ensemble of sparse logistic regression (with second order interactions), lambda mart (with AUC loss), and a restricted Boltzmann machine. To combine them we used sparse logistic regression again with linear weighted features.

It is a 2-layer SVM approach. We designed the method ourselves. We were out of time (i.e. time was spent on the code development) and weren't able to make a competitive submission by the deadline. But, we hope to use one of the datasets as the test set and potentially write up a submission for ECML.

Deep CNN for Time Series Correlation Measurement The solution is a deep convolutional neural network and is a novelty in terms of application of CNN to time series correlation measurement. It takes as input two fragments of activity recordings (derivative) and detects patterns indicating correlations between them. One significant improvement was developed by providing one additional row to these two - an average brain network activity increase in each particular timeframe, allowing the CNN to learn the level of noise and different behavior depending on the network activity. Moreover, the recordings are firstly filtered based on optimal threshold of minimum activity increase in a given time frame, keeping fragments of high activity and decreasing the input size to around 1% of the initial input length. Training uses the same number of positive and negative examples, total 1.2 million. The initial ideas of network structure were based on Lenet5 Network for digit classification (by Y. LeCun). The network behavior has been inspected deeply, including exporting activation values of Theano hidden layer units and classifying them with SVM (libSVM, libLinear), allowing to adopt the network architecture specifically for the task of correlation detection. The code was developed in Python with Theano and is designed for running on a GPU.

My approach was to use ICA-based LiNGAM

References

asd

(1)Babadi, B., Yuste, R., Paninski, L., Vogelstein, J. T., Packer, A. M., Machado, T. A., & Sippy,

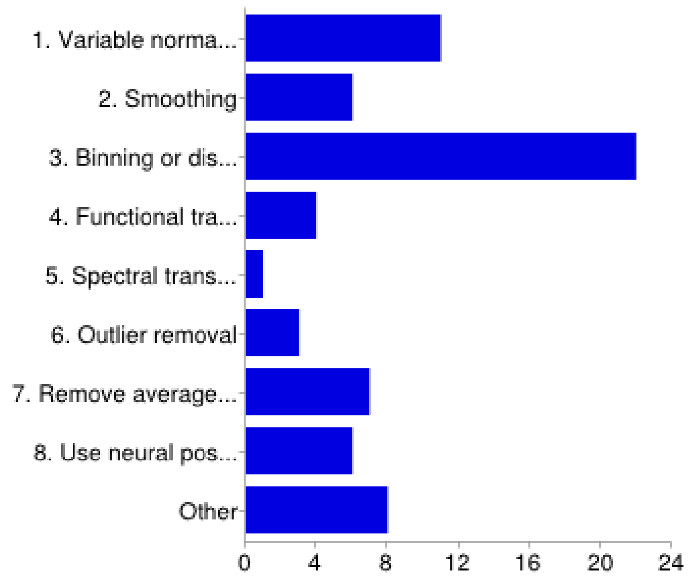
T. (2013). Fast Nonnegative Deconvolution for Spike Train. J. Neurosci, 33(23), 9813-9830. 2))Feizi, S., Marbach, D., Médard, M., & Kellis, M. (2013). Network deconvolution as a general method to distinguish direct dependencies in networks. Nature biotechnology.
Bontempi, Meyer Causal filter selection , ICML 2010
Since I am not in academia any more, I do not plan to publish my work. I am not aware of further relevant methodical references in addition to the ones above - even though I am sure they exist. A very relevant technical reference is the homepage of the scikit-learn project, http://scikit-learn.org. scikit-learn has served as the basis for all statistical computations, i.e. logistic regression and gradient boosting.
http://www.cs.helsinki.fi/group/neuroinf/lingam/
It is such a crude model that it hardly seems worth publishing in it's current form. Also, I'm not familiar with the literature in this field, and therefore cannot provide references at this time.
1. Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32. 2. Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." The Journal of Machine Learning Research 12 (2011): 2825-2830. 3. Czarnecki, Wojciech M. and Jozefowicz, Rafal “Neuronal Connectivity Reconstruction from Calcium Imaging Signal using Random Forest with Topological Features” in preparation for ECML-PKDD 2014 workshop
We plan to write a paper for the ECML Workshop explaining in details our solution. References - -------------- [1] De La Fuente, A., Bing, N., Hoeschele, I., & Mendes, P. (2004). Discovery of meaningful associations in genomic data using partial correlation coefficients. Bioinformatics, 20(18), 3565-3574. [2] Stetter, O., Battaglia, D., Soriano, J., & Geisel, T. (2012). Model-free reconstruction of excitatory neuronal connectivity from calcium imaging signals. PLoS computational biology, 8(8), e1002653.
Fan, Rong-En, et al. "LIBLINEAR: A library for large linear classification." The Journal of Machine Learning Research 9 (2008): 1871-1874.
Vogelstein, J. T., Packer, A. M., Machado, T. a, Sippy, T., Babadi, B., Yuste, R., & Paninski, L. (2010). Fast nonnegative deconvolution for spike train inference from population calcium imaging. Journal of Neurophysiology, 104(6), 3691–704. doi:10.1152/jn.01073.2009 Stetter, O., Battaglia, D., Soriano, J., & Geisel, T. (2012). Model-Free Reconstruction of Excitatory Neuronal Connectivity from Calcium Imaging Signals. PLoS Computational Biology, 8(8), e1002653. doi:10.1371/journal.pcbi.1002653

Supplementary on-line material

https://docs.google.com/file/d/0B98iXdxQRL0TUHVYR3Q3UFpZak0
asd
https://www.flickr.com/photos/124316065@N05/14111659731/sizes/o/in/photostream/
http://www.cs.helsinki.fi/group/neuroinf/lingam/
NA - maybe later
https://github.com/asutera/kaggle-connectomics

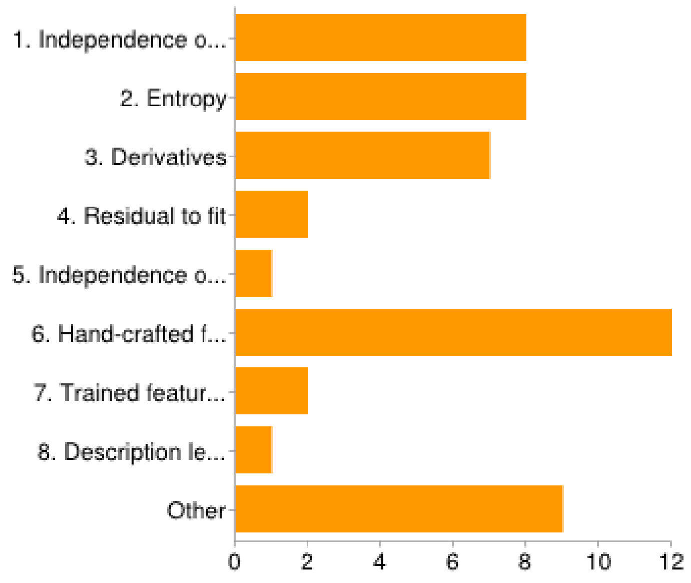
Algorithms of network reconstruction

Preprocessing of fluorescence signals



1. Variable normalization	**11**	35%
2. Smoothing	**6**	19%
3. Binning or discretization	**22**	71%
4. Functional transform (e.g. log)	**4**	13%
5. Spectral transform	**1**	3%
6. Outlier removal	**3**	10%
7. Remove average neural activity	**7**	23%
8. Use neural positions	**6**	19%
Other	**8**	26%

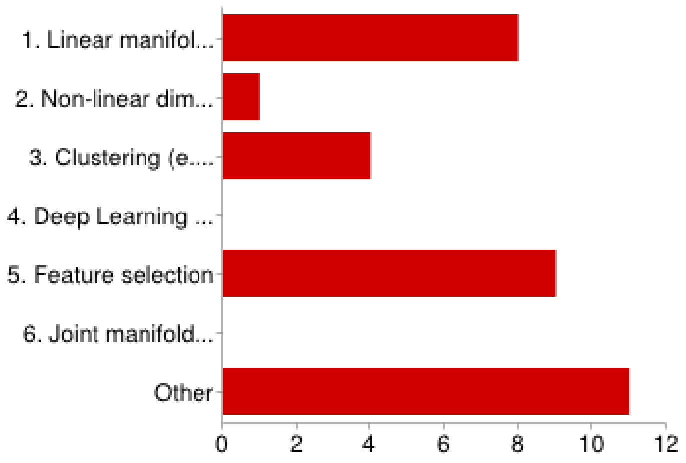
Feature extraction



1. Independence of variables	**8**	26%
2. Entropy	**8**	26%
3. Derivatives	**7**	23%
4. Residual to fit	**2**	6%
5. Independence of input and residual	**1**	3%

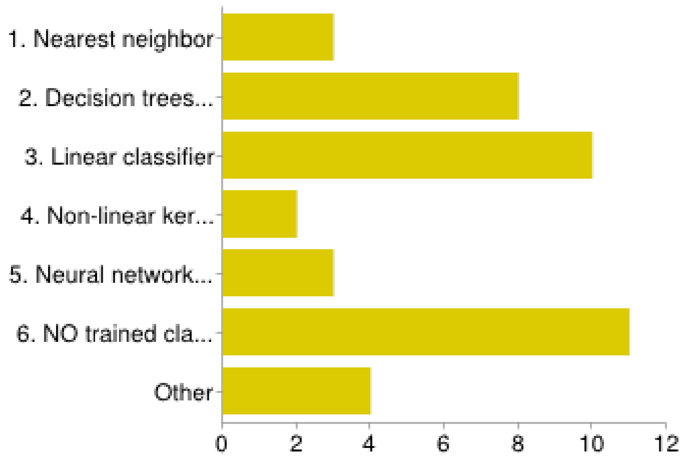
6. Hand-crafted features	**12**	39%
7. Trained feature extractors	**2**	6%
8. Description length or complexity of model	**1**	3%
Other	**9**	29%

Dimensionality reduction



1. Linear manifold transformations (e.g. factor analysis, PCA, ICA)	**8**	26%
2. Non-linear dimensionality reduction (e.g. KPCA, MDS, LLE, Laplacian Eigenmaps, Kohonen maps)	**1**	3%
3. Clustering (e.g. K-means, hierarchical clustering)	**4**	13%
4. Deep Learning (e.g. stacks of auto-encoders, stacks of RBMs)	**0**	0%
5. Feature selection	**9**	29%
6. Joint manifold data fusion	**0**	0%
Other	**11**	35%

Classifier

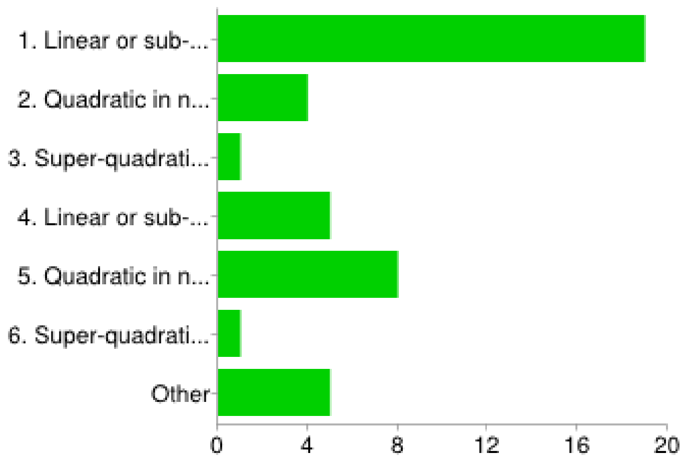


1. Nearest neighbor	**3**	10%
2. Decision trees or random forests	**8**	26%
3. Linear classifier	**10**	32%
4. Non-linear kernel method	**2**	6%
5. Neural networks or deep learning	**3**	10%
6. NO trained classifier	**11**	35%

Other	**4**	13%

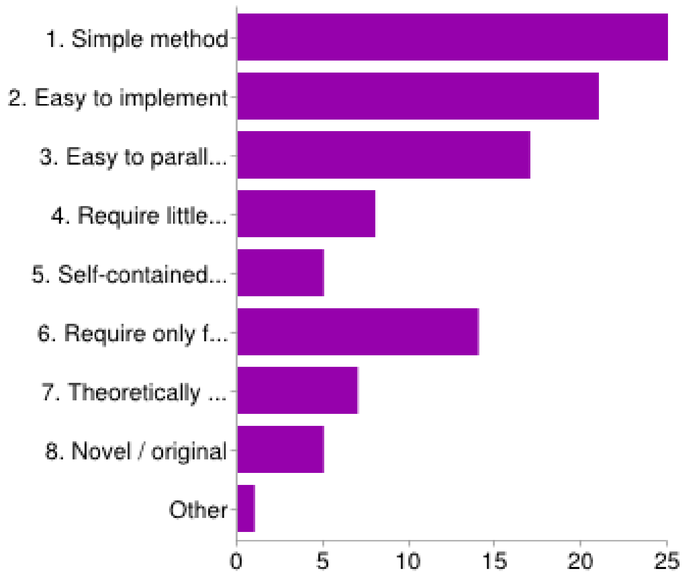
Method advantages

Algorithmic complexity at TEST time



1. Linear or sub-linear in number of test samples per neuron	**19**	61%
2. Quadratic in number of test samples per neuron	**4**	13%
3. Super-quadratic in number of test samples per neuron	**1**	3%
4. Linear or sub-linear in number of neurons in test data	**5**	16%
5. Quadratic in number of neurons in test data	**8**	26%
6. Super-quadratic in number of neurons in test data	**1**	3%
Other	**5**	16%

Qualitative advantages



1. Simple method	**25**	81%
2. Easy to implement	**21**	68%
3. Easy to parallelize	**17**	55%

4. Require little memory	**8**	26%
5. Self-contained (does not rely on third party libraries)	**5**	16%
6. Require only freeware libraries	**14**	45%
7. Theoretically motivated	**7**	23%
8. Novel / original	**5**	16%
Other	**1**	3%

Comparison with other methods

Our method does not require a training and is very simple in comparison to other methods. Nevertheless, the biggest contribution of our method is probably in the pre-processing of the data.

The inverse cross variance was the most critical element. A score of around .924 was achievable with only a few lines of code and something that runs quite quickly!

Logistic regression weights as basic features strike a good balance between computational load and incorporating correlations between different neurons, i.e. elements of the graph structure. In fact, even without much optimization, scores from logistic regression tended to perform significantly better (approximately AUC=.92 compared to AUC=.90) than the two- or three-neuron based features such as mutual infomation and conditional mutual information that I had tried before. Classification time is still reasonably fast, taking about 10min on a laptop to compute the score for 1000 neurons. By transforming the binarized signals that are used as predictors, temporal information - at least over a few timesteps - can be easily integrated. Optimization of the corresponding parametrization is very time-consuming (on the order of days on a single processor), but can be parallelized trivially. Prediction time is not affected. The usage of Gradient Boosting with random undersampling performed much better than all the other classifiers in scikit-learn, training time is short (about one hour), and generalization is good. The fact that it is a tree-based classifier allows to incorporate all kinds of additional features without having to care about continuous vs categorical, normalization etc.

My current thinking is that there was at least one aspect of the challenge format (the inclusion of a large number (~10%?) of blocked-response neurons in the simulation, perhaps) that effectively "penalized" approaches like GTE, i.e. methods that provide a directional rather than merely correlative indication of connection. I chose my final model only because it gave the best leaderboard score I had gotten up to that point.

asd

Linear correlation only can produce result with comparable performance with GTE if properly regularized. CTE give good performance in different dynamical regimes, while the GTE proposed by the organizers only give relatively good performance in certain dynamical regime. CTE is fast. With some slight modification to the MATLAB GTE code provided by the organizer, it only takes less than 40 min to run a sample with 1,000 neurons on a regular Desktop computer.

I would never use the method which I implemented in practice -- the GTE approach in Setter et. al. is much more elegant and probably more reliable on real data as it does not involve any black-box learning!

The gaussian kernel.

As mentioned briefly in the description using topological features proved to be very useful and help a lot in classifying connections.

I've tried to extract many features and used a stack of highly robust classifiers. In the end the best solution was a simple procedure which used theoretical knowledge (Fast-oopsi) and a very smart algorithm for the reduction of non-direct edges (Network deconvolution)

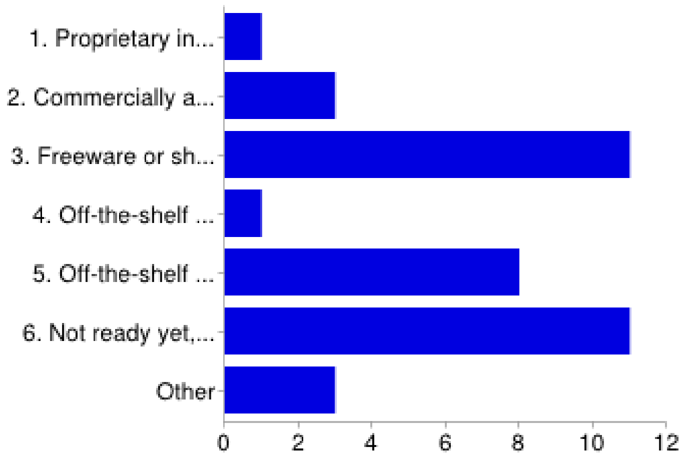
The initial idea was to take fragments of activity recordings (discretized to 4 values) and compute the probability that cells having these exact pairs of fragments are connected. To handle storing different time series probabilities, the length of one fragment was set to 6, resulting in 4^(2 * 6) combinations. The computations of subsequent pairs values were done in constant time by shifting the value and were executed in C. When scores were summed, that gave the AUC score of 90.0%, higher than provided benchmarks. In the CNN filtering active parts of recordings, providing information about network activity in the extra row, deep analysis of network behaviour and using proper activation functions in all layers led to substantial increase in accuracy.

I think the critical element was to find an adequate signal base to represent data. My approach find connectivity using ICA, which represent data with independent signals. After that, I compute the connectivity.

The main reason we were able to improve on GTE as presented in (Stetter et al. 2012) is that we were able to remove (or at least reduce) spurious correlations between pairs of nearby neurons arising due to light scattering. Stetter et al required a Markov order of 2 in order to obtain good performance from GTE in the presence of light scattering, whereas we were able to use a Markov order of 1, allowing us to achieve better estimation of the conditional probabilities (and thus the transfer entropy) given the available sample size. We also benefited to a lesser extent from using fast non-negative deconvolution to achieve a better estimate of the spike trains from the deblurred fluorescence traces.

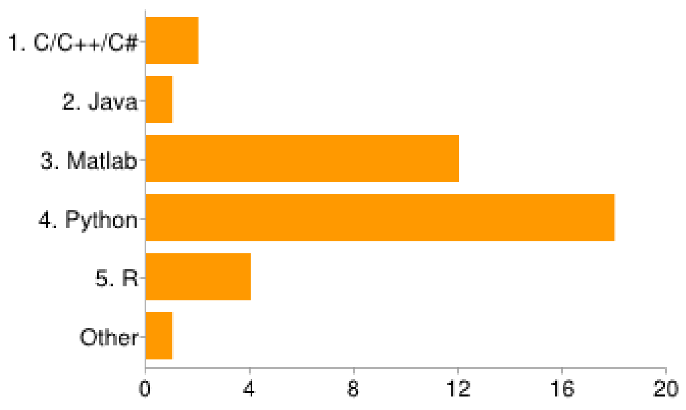
Software implementation

Availability



1. Proprietary in house software	**1**	3%
2. Commercially available in house software	**3**	10%
3. Freeware or shareware in house software	**11**	35%
4. Off-the-shelf third party commercial software	**1**	3%
5. Off-the-shelf third party freeware or shareware	**8**	26%
6. Not ready yet, but may share later	**11**	35%
Other	**3**	10%

Language



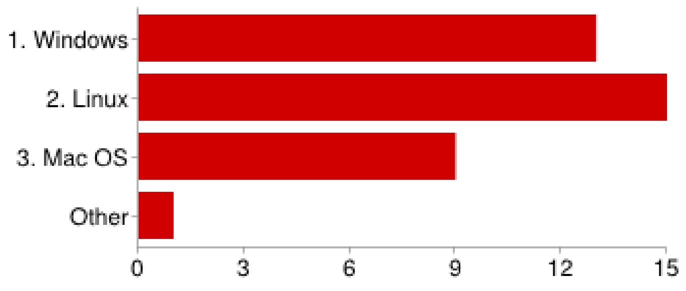
1. C/C++/C#	**2**	6%
2. Java	**1**	3%
3. Matlab	**12**	39%
4. Python	**18**	58%
5. R	**4**	13%
Other	**1**	3%

Details on software implementation

Implementation is completely in Python (2.7). In addition to the standard libraries, only pandas (for csv reading), numpy/scipy (mainly linear algebra and sparse matrices) and scikit-learn are used. For my own implementation I have used the anaconda distribution of python.
fast-oopsi implementation - https://github.com/jovo/oopsi (matlab - but after the competetion was over another competitor has published a python implementation - https://github.com/liubenyuan/py-oopsi) Network deconvolution was converted from the original matlab code provided by the authors to python
Activity values are discretized to 0/1, and only the time indices of '1' values are stored in sets for each neuron. This accelerates the correlation calculations, and reduces memory consumption.
The following programs and packages were used for the contest: - Python 2.7 - NumPy >= 1.6.2 - SciPy >= 0.11 - scikit-learn >= 0.14 with appropriate blas and lapack binding such as MKL, accelerate or ATLAS. In order to test the code, we recommend you to use the Anaconda python distribution (https://store.continuum.io/cshop/anaconda/).
Compiled various GTE-related algorithms in FORTRAN and linked them to R. These routines not utilized in final submission, but could be useful for future work.
Python and Theano.
Scipy and sklearn
Written in python using numpy, scipy and scikit-learn with critical paths rewritten in cython for speed.
The bulk of the data processing pipeline was written in Python, using numpy, scipy, and scikit-learn. We wrote our own implementations of fast non-negative deconvolution and generalized transfer entropy, which were partly coded in Cython for speed. For fast non-negative deconvolution we also greatly speeded up the computation of the Hessian matrix by wrapping the dgtsv solver for tridiagonal systems from the LAPACK Fortran library. We hope to make both of these packages publicly available very soon.

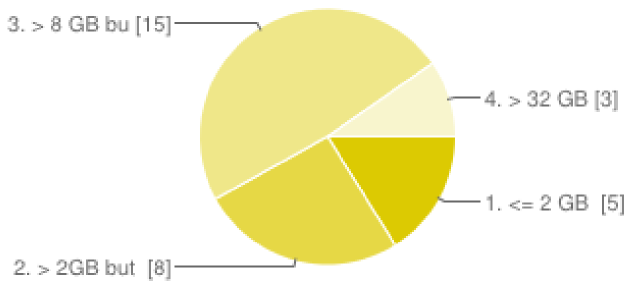
Hardware implementation

Platform



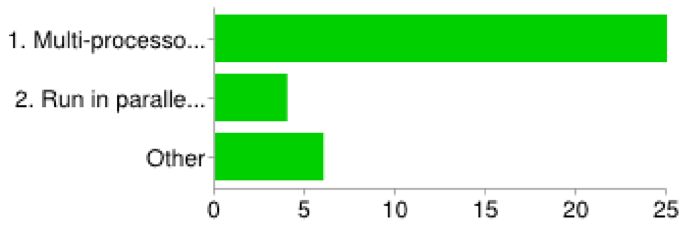
1. Windows	**13**	42%
2. Linux	**15**	48%
3. Mac OS	**9**	29%
Other	**1**	3%

Memory



1. <= 2 GB	**5**	16%
2. > 2GB but <= 8 GB	**8**	26%
3. > 8 GB but <= 32 GB	**15**	48%
4. > 32 GB	**3**	10%

Parallelism



1. Multi-processor machine	**25**	81%
2. Run in parallel different algorithms on different machines	**4**	13%
Other	**6**	19%

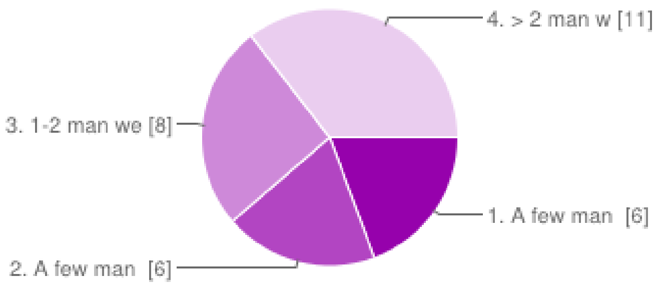
Code URL

No yet available

https://github.com/asutera/kaggle-connectomics

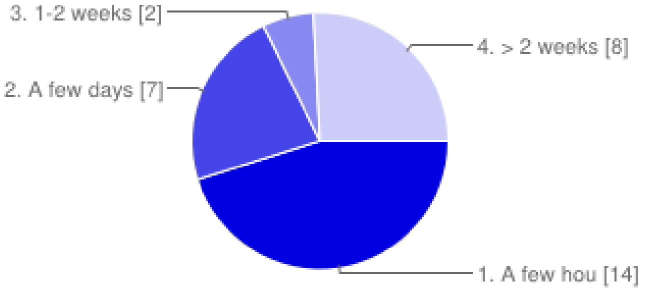
Development effort

Total human effort



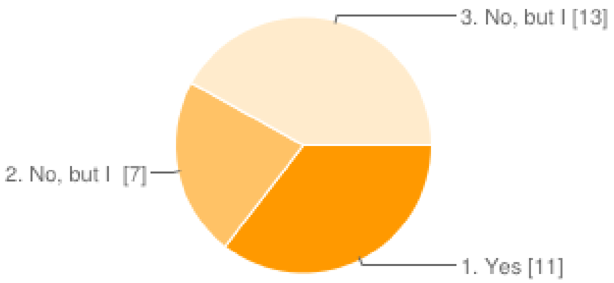
1. A few man hours	**6**	19%
2. A few man days	**6**	19%
3. 1-2 man weeks	**8**	26%
4. > 2 man weeks	**11**	35%

Total machine effort



1. A few hours	**14**	45%
2. A few days	**7**	23%
3. 1-2 weeks	**2**	6%
4. > 2 weeks	**8**	26%

Challenge duration OK?



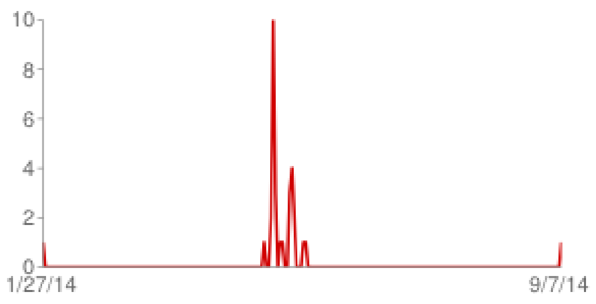
1. Yes	**11**	35%
2. No, but I cannot spend more time	**7**	23%
3. No, but I wish to enter round 2 of the challenge	**13**	42%

Final evaluation time (hours)

333
~40 min
0.5
~24
With GTE a long time, without GTE (sacrificing a little error) about half a day
2
1

about 10
5
around one hours with a new test dataset on a regular Desktop PC
12
~10 minutes
10
0.15 hours (approximately)
2-3 hours
< 1 hr
5h
To train classifiers on the 4 training sets took my laptop ~4 hours. Then to actually run predictions on the valid/test tests took 0.1 hours.
48
Single feature generation takes between 1 and 30minutes. On one cpu generating all of them should finish within 24-48 hours but it is fully parallelizable. These needed to be generated only a single time and we did it throughout the competition. Once generation is finished, the evaluation takes about a minute.
I set it to run 15h to evaluate final set. I did 8 runs in the same time with different seeds and averaged to increase accuracy a little.
50

Number of daily responses


10
8
6
4
2
0
1/27/14
9/7/14