# 8 Integrative modeling (TL, ||)

## 8.1 S (TL, ||) Genotype - gene expression/enhancers - modules - phenotype (JW)

We integrate data of the kinds described above into a single model connecting genotype, functional genomics and phenotype data from PsychENCODE in the Prefrontal Cortex. We build separate models for the phenotypes Schizophrenia (SCZ), Bipolar disorder (BIP), Autism spectrum disorder (ASD), age (AGE), gender (GEN) and reported ethnicity (ETH). For each phenotype, we created 10 balanced train / test splits as described below, and we report the performance of all models averaged across these 10 splits of the data. For the disease conditions, these splits contain equal numbers of cases and controls, while for age, gender and ethnicity, only control samples are used. As inputs to the model during training, we use the imputed genotypes; intermediate phenotype data including gene expression, enhancer h3k27ac activation levels, cell fraction estimates, and co-expression module mean expression; and high-level phenotype data corresponding to the categories above. Gene expression and enhancer activation data was quantile normalized, as for the QTL calculations. Also, for the cRBM, cDBM and DSPN models, all functional genomics data was binarized by thresholding at the median value (per gene/enhancer/cell-type/module). Further, DSPN model connectivity was constrained by using the estimated eQTLs, cQTLs and fQTLs, along with the GRN TF-gene and enhancer-gene linkages estimated in the elastic net analysis.

8.1.1 Balanced Datasets

We first describe how the balanced datasets are created for SCZ, and then describe how the balanced datasets are created for the other high-level phenotypes using a similar process with small modifications. For SCZ, we divide the PEC data into subsets, each containing samples from a common assay (BipSeq, brainGVEX, CMC, CMC-HBCC, Libd, UCLA-ASD, Yale-ASD or GTeX-DFC), the same gender (M or F), the same ethnicity (Caucasian (CC) or African American (AA), to which most samples belonged), and the same age range (1-9, 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90+). For each subset, we found all SCZ and CTR samples within that subset, with counts and for the number of cases and controls respectively. We then sampled uniformly without replacement SCZ samples and CTR samples from the subset to add to a ‘pool’ of samples for the current data split. After having done this for all such subsets so that the pool contains SCZ and CTR samples, we partition the case samples randomly into groups of size and for training and testing respectively (), and do likewise to add equal numbers () of controls to each partition. We repeat the whole process 10 times to generate 10 data splits; the above process ensures that each training and test partition contains a 50/50 split of SCZ/CTR samples, and additionally that the distribution of covariates (assay, gender, ethnicity and age) is approximately the same for cases and controls in the training and testing partitions.

Exactly the same method is used to create balanced data splits for BIP. For ASD, due to the limited number of cases, we set , and balance only for assay and gender (not for ethnicity and age range). For the non-disease phenotypes (AGE, GEN and ETH), a similar method is used but with the following modifications. Here, we use CTR samples only, and split the PEC data into subsets containing samples from a common assay, and which are matched on all covariates as above except the high-level phenotype being modeled. Then, equal numbers of samples are randomly selected for each binary value of the modeled phenotype to be added to the training/testing partitions (respectively and for training and testing as above); for GEN the binary values are M/F, for ETH they are CC/AA, and for AGE we binarize the trait as 0/1 such that 1 indicates that a sample is older than the median age of 51 (NB the median age binarization is used only when AGE is the modeled phenotype; for all other phenotypes age is balanced using the decade age bins as above). The above method generates 10 data splits each of the following sizes (training/testing): SCZ (640/70); BIP (170/18); ASD (50/12); AGE (244/26); ETH (284/30); GEN (312/34).

## 8.2 S (TL, ||) Deep learning model for predicting brain genotype-phenotype (JW)

## 8.2.1 Logistic regression (LR)

We train LR models to predict a binary phenotypes from a single level of predictors (either genotype or an intermediate phenotype). The model has the form:

(1)

where is the phenotype, is a vector of predictors, is a weight vector, is the bias term, and is the logistic function, . Since training and test sets are both balanced, for a test sample we use the predictor , where is the Iverson bracket, which is 1 if *a* is true, and 0 otherwise.

For each data split, we initially perform feature selection by calculating the correlation of each predictor with the high-level phenotype:

(2)

,

where is the phenotype of the *i*’th training sample, is the value of the *j*’th predictor at the *i*’th training sample, and is the Pearson correlation function, . To perform feature selection, we rank the predictors by the absolute value of in descending order for a given training split, and include only predictors in the model for that data split. We learn two LR models for each phenotype, the first using the imputed genotypes at the eSNPs as predictors, and the second using PFC gene expression levels (transcriptome) as predictors. We set and for the genotype and transcriptome models respectively. For optimization, we use the Matlab Statistics and Machine Learning toolbox (glmfit).

8.2.2 Conditional Restricted Boltzmann Machine (cRBM)

A Restricted Boltzmann Machine (RBM) models the joint distribution of a set of visible and hidden units; we will denote the visible units as andcorresponding to the intermediate and high-level phenotypes respectively, and the hidden units as , all of which are binary variables (multivariate in the case of and).An RBM has the form , where is a normalizing partition function, and is the RBM energy function, which has the form , where is a matrix of interaction weights between the visible and hidden units, and andare the visible and hidden bias terms respectively. A Conditional RBM (cRBM) models the conditional distribution of a set of visible and hidden units on a further set of conditioning (visible) units (see \cite{mnih\_uai11}), which we will denote , and which are assumed to be discrete:

(3)

,

where is a matrix of interaction weights between the conditioning and visible units (which are restricted here to exclude interactions involving , and hence model only dependencies between genotype and phenotype which are mediated by the intermediate phenotypes ).

Both the RBM and cRBM may be trained using Contrastive Divergence (CD). In the case of the cRBM, CD finds an approximate gradient to the conditional log-likelihood of the training data:

(4)

where denotes the expected value of after performing steps of alternating Gibbs sampling, starting with the visible units fixed to the training data (see \cite{hinton\_12} for the RBM case). Approximate gradients for interactions involving and and the bias terms may be found similarly by estimating the expected statistics for , , and after one step of alternating Gibbs sampling. The step size for the change in at iteration , , may then be calculated as:

(5)

where is a momentum parameter, is the learning rate, and is a weight cost to encourage sparsity. At each iteration, we evaluate Eq. 5 using a subset of the training samples (a mini-batch), hence performing stochastic gradient descent (SGD). We cycle once through the training data in disjoint mini-batches to form an epoch, and use early stopping after epochs to control for overfitting.

Given a test sample, we wish to predict given and (or alone for imputation based inference, see below). This can be achieved by maximizing the conditional probability of and given **,** or equivalently minimizing the free-energy (see \cite{hinton\_12} for the RBM case):

(6)

We use the 10 balanced data split above to train a series of models for each phenotype. We initially perform feature selection (for each data split) using the method in Eq. 2 to identify a subset of genes to include as transcriptome predictors in (setting ), and include all eSNPs associated with these genes in . We also enforce sparsity on the matrix during training, so that only connections supported by eQTLs are allowed to be non-zero. Further, we set (the number of hidden nodes), , , , and used mini-batches of size 61, 10, 17, 71, 39 and 64 for AGE, ASD, BIP, ETH, GEN and SCZ models respectively. Performance for each phenotype is calculated as an average across data splits for the accuracy of a model on its corresponding test partition.

8.2.3 Conditional Deep Boltzmann Machine (cDBM)

A Deep Boltzmann Machine (DBM) may be defined as in \cite{salakhutdinov09} as a Boltzmann machine with additional structure such that it can be viewed as a stack of RBMs. The model with two hidden layers has the form: , where is a normalizing partition function, and is the DBM energy function, which has the form . Here, **, ,** are matrices of interaction weights between the visible and first-layer hidden units, the first and second layer hidden units, and the ‘labels’ and second-layer hidden units respectively. For the DBM, we write as a vector, since for convenience we assume the class variables (high-level phenotypes) are represented using one-of- encoding (i.e. for a binary trait, either or for the two classes), and we write for a single vector combining all the bias terms.

As for the cRBM, we can use a family of DBMs to model a conditional distribution which depends on a further set of variables, **.** This is equivalent to converting the DBM from a Markov Random Field (MRF) into a Conditional Random Field (CRF, see \cite{koller09}). We can thus define a conditional DBM analogously to the cRBM:

(7)

,

The cDBM can be trained by adapting the Persistent Markov Chain Monte Carlo algorithm used in \cite{salakhutdinov09}. In this approach, following a pre-training phase which uses CD to train adjacent layers as RBMs, the weights for the whole network are optimized jointly by approximating the gradient to the full data log-likelihood of the model. For the cDBM, we can write the approximation as:

(8)

where for convenience we show only the gradient for a weight in matrix . The first term uses a mean-field approximation to evaluate the conditional expectation of when andare clamped to their observed values (due to this clamping, the unimodal form of the mean-field distribution is expected to hold approximately). Mean-field updates in the cDBM may be calculated straightforwardly by incorporating terms involving into the energy. The second term approximates the model statistics with unclamped; in the case of the DBM a set of persistent Markov Chains are maintained for this purpose, each tracking the trajectory of a ‘fantasy particle’ consisting of a joint setting of the model variables ). The fantasy particles make a fixed number of updates at each gradient iteration using the current model weight settings, and are not re-initialized (hence ‘persisting’) between gradient updates (each can be thought of as a series of Markov chains with changing parameters, or a single Markov chain over the model variables and weight parameters). The fantasy particles can then be used to estimate the required model expectations for the gradient. A similar approach can be used for the cDBM, only because the required term in the gradient is now a conditional expectation, it cannot be estimated by calculating expectations over a set of fantasy particles all evolving according to the same Markov process. Rather, a set of fantasy particles is required for each training sample (, each evolving according to a Markov process conditioned on that sample’s value, and the expectation is calculated across the entire collection. Stochastic gradient updates are then made to the weights as in Eq. 5 (substituting for ). Finally, as in \cite{salakhutdinov09} back-propagation can be applied for fine-tuning, and use a single forward pass through the network for prediction. Settings of the parameters above are described in the context of the DPSN in the following section.

8.2.4 Deep Structured Phenotype Network (DSPN)

## We define a Deep Structured Phenotype Network (DSPN) as a conditional Deep Boltzmann Machine, with extra structure added to the visible units to reflect regulatory relationships between various intermediate phenotypes. The general form of the model is:

(9)

,

which is identical to the cDBM, except for the introduction of a matrix of interaction terms between the visible units. However, we require that **,** and have specific forms, such that:

(10)

,

where are (binarized) representations of the gene expression, enhancer activity (h3k27ac level), cell-type fraction and co-expression module net activation respectively; is a sparse matrix where non-zero entries are allowed only between genes having a TF-target relationship determined by the elastic net model; is a sparse matrix where non-zeros are allowed only between enhancers and genes when an enhancer-target link is determined by the elastic net model; and are sparse matrices where non-zero entries are allowed only between a cell-type/co-expression module and the marker-genes/member-genes associated with it respectively; and are sparse matrices with non-zero elements allowed only between SNPs and genes/enhancers/cell-types/modules supported by a QTL linkage. We note that the results of previous analyses (e.g. elastic net and QTL analyses) are used only to establish the sparse structure of the and matrices, but not the actual linkage values of the non-zero entries, which are learned during joint training of the DSPN model (along with the and parameters). In general, we do not expect the magnitudes established independently for these linkages in the previous analyses to relate in a straightforward way to their optimal settings in a joint model, and hence we use only the connectivity structure as prior information during training.

The DSPN model can be trained similarly to the cDBM using persistent MCMC as described above. Mean-field approximate inference and Gibbs sampling steps are straightforwardly adapted to incorporate the additional linkages between the visible units. Because of the dependencies within the visible units, the mean-field and sampling steps cannot be made in parallel for the visible layer unlike the cDBM; for this reason, we choose a random update schedule of the nodes within the visible layer on each iteration, and update all other layers in parallel as before. In principle, the approach described learns a model representing the joint distribution of intermediate and high-level phenotypes conditioned on genotypes, and so can be used for prediction of high-level phenotypes either directly from the intermediate phenotypes, or from the genotype with imputation when the intermediate layers are unobserved. However, we adopt a slightly different training process when the goal is to provide a model for inference with imputed intermediate phenotypes, as described below, to optimize performance for this scenario. We summarize here the parameter settings for the model with direct observations: we perform feature selection as in Eq. 2 for each intermediate phenotype (setting ); additionally, we set and (the number of hidden nodes in layers 1 and 2 respectively), , , , , and use mini-batches of the same sizes as those for the cRBM.

## 8.3 S (TL, ||) Imputation using integrative modeling (JW)

## 8.3.1 Deep Structured Phenotype Network for Imputation (DSPN-Imput)

## To optimize performance for prediction of high-level phenotypes from genotype data with imputation of intermediate phenotypes, we adopt a specialized training process. We assume that during training, we have access to fully observed genotype and intermediate phenotype data. Additionally, we split the training data for each data split evenly into training and validation partitions.

## First, we train logistic regression models independently to predict each intermediate phenotype (e.g. gene expression level, enhancer activation) from the genotype at each of its QTLs using the training partition. We then fix the matrices of the DSPN directly to the coefficients of the logistic regression models, and train and matrices (along with the biases for the visible layer and first hidden layer; primes indicate that these parameters are initial estimates only) by optimizing on the validation partition, while fixing all hidden nodes at the second layer to 0; since we only allow one level of hidden nodes to vary, this model is equivalent to a cRBM (with additional structure on the visible nodes), and hence we use the Contrastive Divergence (Eq. 5) for optimization. Additionally, we perform feature selection at this stage by only including in the model the top proportion of intermediate phenotypes for each respective type as order by their predictive accuracy using the initial logistic predictor. We then use the partial cRBM model over to jointly infer estimated intermediate phenotype data for the validation samples, which we label (we infer by initializing it to the maximum likelihood outputs of the logistic predictors, and performing Gibbs sampling according to the cRBM energy function to refine this estimate). Finally, we train a full DSPN (with still fixed) on the validation data, but optimized using the imputed rather than the original intermediate phenotype data, i.e. using as training samples.

## At test time, we do not make use of the intermediate phenotype data. Instead, we follow a similar path to training, by first imputing the intermediate phenotype data using the partial cRBM model with parameters and (initialized using the individual logistic predictors used to form ). We then treat the imputed phenotype data as fixed, and predict the associated high-level phenotype data from the full DSPN model using a forward pass as described for prediction in the cDBM model. We train the imputation based DSPN model using the same hyper-parameters as for the DSPN above, while setting

## 8.4 S (TL, ||) Variance explained on liability scale and heritability estimation (JW)

We use GCTA to estimate the heritability of SCZ, BIP and ASD from the PEC genotype samples on the liability scale \cite{yang11}. We compare heritability estimates when common SNPs (MAF>0.1) and eSNPs are used as input. We use the top 3 genotype PCs as covariates to control for population structure (as in the eQTL calculations), and apply LD deflation to produce a subset of 50K common SNPs and 10K eSNPs to serve as marker SNP inputs to GCTA.

To covert predictive performance of all models onto the liability scale, we use the following conversion due to Falconer (see \cite{isc09} and \cite{falconer96}):

(11)

Here, is the variance explained on the liability scale, is the probability the model predicts a genotype to be a case, GRR is the genotype relative risk, and , where is the disease prevalence, and the height of a standard normal distribution when the cumulative distribution has height . Letting be the true negatives, false negatives, false positives and true positives respectively for a given model on test data, we estimate , and . We set for SCZ, BIP and ASD respectively.

## 8.5 S (TL, ||) DSPN enrichment analysis for prioritized modules and higher-order groupings (JW)

To provide interpretation of the DSPN model we develop a multilevel prioritization scheme, which, given a node of interest and a lower ‘projection layer’, defines positive and negative subsets of nodes on the projection layer which are most ‘important’ in influencing the value of the node of interest. In our analysis, we take the node of interest to be either a high-level trait (e.g. SCZ), or a hidden-layer node, and the projection layer to be an intermediate phenotype; we then use the prioritized subsets either to look for intermediate phenotypes prioritized for a given trait, or to functionally annotate hidden-layer nodes by looking for functionally enriched categories in the prioritized subsets.

In general, we assume we have a neural network with layers , , with the lowest (input) layer and the highest (output) layer. We fix a node of interest on layer , , and a ‘branching factor’ , which will determine the maximum size of the prioritized sets associated with . Given these, we recursively define the positive and negative sets and associated with for all . We start by defining and . Then, for all :

(12)

where we define the sets for as and , where the function returns the rank of when the nodes of layer are ranked in descending order by the network weights , and returns the rank when the nodes are ranked in ascending order by the same weights. We note that and may contain common elements (i.e. nodes that contribute both positively and negatively to variation in a higher-level node).

To find prioritized modules for a given trait, we fix the ‘projection layer’ to be the co-expression module sublayer in the DSPN (), and find the sets and when is set to the output trait node. We repeat this analysis for models trained on the 10 splits of the data for the given trait, generating 10 positive and negative projected sets. For module , we then calculate the counts , where is the positive projected set from the model trained on data split , and . For our final list of positive and negative prioritized modules we use and respectively. The threshold is set such that, under a null distribution where the network weights are sampled from a standard normal distribution and the same branching factor is used. We set , and evaluate using 10,000 simulations. Setting , we found that this implied an estimate of , and generated   positive and negative prioritized modules per trait (out of ).

To annotate ‘typical’ ancestor nodes of module at layers and in the DSPN (hidden layers and respectively), for each data split we find nodes and such that forms the ‘best path’ from module to the trait output node in the sense that it minimizes the score:

(13)

across all ‘positive’ paths, meaning:

,

and ties are broken arbitrarily (a similar annotation can be made for negative paths by placing on the RHS of Eq. 13). Writing for the nodes on the best path from module in the model from data split , we evaluate the counts for all modules, and , where we write for the positive projected set at level for data split when we set the node of interest . We then evaluate and where is defined as above, and annotate a typical (positive) ancestor of at layer (respectively ) by finding the functional annotations enriched in the gene-set formed by taking the union of the co-expression modules in (respectively ).

We perform functional enrichment analysis using the R package ‘clusterProfiler’ \cite{yu12} using KEGG pathway annotations, and setting the p-value and q-value cutoffs to 0.05 and 0.1 respectively. Further, we perform enrichment analysis for the cell-type marker genes corresponding to the cell-types used in our single-cell deconvolution analysis. Here, we threshold the marker gene expression matrix for each gene independently at its 0.75 quantile value to define a collection of subsets of marker genes for each cell-type. We test for enrichment of cell type markers using the hypergeometric test with a p-value cutoff of 0.001. Finally, we also compare the modules prioritized for our SCZ model using the above approach with those prioritized using a gradient-based approach, following \cite{simonyan13}, where the magnitude of the gradient of the response of a node of interest (in our case, the trait node responses across the training set) is use to prioritize salient input nodes (modules). We provide the results of this analysis in our supplementary data, but found it to exhibit a strong bias towards prioritizing smaller modules, which may be due to the underestimation of the contribution of saturated nodes in gradient approaches (see \cite{shrikumar17}, which attempts to circumvent these problems, but requires definition of a ‘reference’ state which is unclear in our model), causing us to prefer the prioritization scheme developed above, in which we did not observe such a bias.

**References**

{mnih\_uai11}: [Volodymyr Mnih](https://arxiv.org/find/cs/1/au%3A%2BMnih_V/0/1/0/all/0/1), [Hugo Larochelle](https://arxiv.org/find/cs/1/au%3A%2BLarochelle_H/0/1/0/all/0/1), [Geoffrey E. Hinton](https://arxiv.org/find/cs/1/au%3A%2BHinton_G/0/1/0/all/0/1), Conditional Restricted Boltzmann Machines for Structured Output Prediction. Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, 2011, Pages 514-522.

{hinton\_12}: Hinton G.E. (2012) A Practical Guide to Training Restricted Boltzmann Machines. In: Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg

{salakhudinov09}: R. Salakhutdinov and G. Hinton. Deep Boltzmann Machines. *AISTATS*, 2009.

{koller09}: D. Koller and N. Friedmann. *Probabilistic Graphical Models*, 2009.

{yang11}: Yang J, Lee SH, Goddard ME, Visscher PM. GCTA: a tool for genome-wide complex trait analysis. The American Journal of Human Genetics. 2011 Jan 7;88(1):76-82.

{isc09}: International Schizophrenia Consortium. Common polygenic variation contributes to risk of schizophrenia and bipolar disorder. Nature. 2009 Aug;460(7256):748.

{falconer96}: Falconer DS & MacKay TFC (1996) Introduction to Quantitative Genetics, Ed 4. Longmans Green, Harlow, Essex, UK.

{yu12}: Yu G, Wang L, Han Y and He Q (2012). “clusterProfiler: an R package for comparing biological themes among gene clusters.” OMICS: A Journal of Integrative Biology, 16(5), pp. 284-287.

{simonyan13}: Simonyan, Karen, Vedaldi, Andrea, and Zisserman, Andrew. Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034, 2013.

{shrikumar17}: Shrikumar A, Greenside P, Kundaje A. Learning important features through propagating activation differences. arXiv preprint arXiv:1704.02685. 2017 Apr 10.