Supplementary material for "Circular inference in schizophrenia"

Renaud Jardri, Sophie Deneve

This supplementary material provides the mathematical derivations for the equations in the main text.

1 The belief propagation (or sum-product) algorithm

The BP algorithm is considered here in its most general form. Less mathematically-minded readers could skip this part and go directly to the next sections, where we present a simpler version of BP only for pairwise graphs.

All hierarchical causal networks can be expressed as a "factor graph". Supplementary figure S1 illustrates how different types of causal models can be translated into factor graphs. A factor graph contains two types of nodes: variable nodes represented by lower case indices, i, j and factor nodes represented by upper case indices I, J, with edges between the variable nodes and the factor nodes in which they appear. As its name indicates, such graphical model represents the factorization of a joint probability distribution of several variables (represented here by a vector $\mathbf{x}$) into a product of smaller factors:

$$p(\mathbf{x}) = \prod_I f_I(\mathbf{x}_{N_I}) \tag{1}$$

where $\mathbf{x}_{N_I}$ is the subset of variables belonging to the I^{th} factor. Whenever a variable i belongs to N_I, there is an undirected edge between variable node i and factor node I. Nodes and edges form a bi-partite graph representing the relation of conditional independence between the variables (see examples shown in fig S1).

The BP algorithm works by passing messages along the edges between the nodes. There are two types of messages: Messages going from variables to nodes and messages going from nodes to variables. Thus, $\mu_{j\to I}(x_j)$ is the message sent from variable node j to factor node I. $\mu_{I\to i}(x_i)$ is the message sent from factor node I to variable node i.

The sum-product algorithm computes the new messages μ's recurrently as follow:

$$\mu'_{j\to I}(x_j) = \prod_{J\in N_j\setminus\{I\}} \mu_{J\to j}(x_j) \tag{2}$$

$$\mu'_{I\to i}(x_i) = \sum_{x_{N_I\setminus\{i\}}} f_I(x_{N_I}) \prod_{j\in N_I\setminus\{i\}} \mu_{j\to I}. \tag{3}$$

where N_j is the set of neighboring (factor) nodes to j. If $N_I \setminus \{i\}$ is empty, then $\mu_{I\to i}(x_i)$ is set to the uniform distribution.

The belief associated with variable node i is the product of all messages converging to it, normalized to sum to one:

$$b_i(x_i) = \frac{1}{Z} \prod_{I \in N_i} \mu_{I \to i}(x_i) \tag{4}$$

Where Z represent a normalization term (independent of x_i) which we will largely ignore here. By abuse of notation, we will call all normalizing terms "Z".

Evidence about any particular variable i in the graph (e.g. priors or likelihoods) can be entered in the form of a constant message sent from a factor node only connected to i. In factor graphs without loops, the algorithm converges to the posterior probabilities of the variables given the entered evidence, as long as messages circulated at least once in each direction in each link.

In the paper, we draw an analogy with neural processing in recurrent, hierarchical networks. To do so, it is useful to see these equations in a different form. Thus, we define log-messages as $M_{I \to i}(x_i) = \log(\mu_{I \to i}(x_i))$ and log-beliefs as $B_i(x_i) = \log(b_i(x_i)) + \log(Z)$. Beliefs can then be simply obtained by exponentiating $B_i(x_i)$ and normalizing to sum to 1. The update equations can be written as:

$$B_i(x_i) = \sum_{I \in N_i} M_{I \to i} \tag{5}$$

$$M'_{I \to i}(x_i) = \log \left(\sum_{x_{N_I \setminus \{i\}}} f_I(x_{N_I}) \prod_{j \in N_I \setminus \{i\}} \exp\left(B_j(x_j) - M_{I \to j}\right) \right). \tag{6}$$

Which we can rewrite in terms of "interaction functions" $W_{Ii}()$ between factor I and variable i as follow:

$$B_i(x_i) = \sum_{I \in N_i} M_{I \to i} \tag{7}$$

$$M'_{I \to i}(x_i) = W_{Ii}\left([B_j(x_j) - M_{I \to j}]_{j \in N_I}\right). \tag{8}$$

Thus, the message sent from I to i is computed from the beliefs of all nodes j connected to I *corrected by* the messages previously sent from I to j. In practise, this equation can run into numerical problems very quickly (i.e. log messages and beliefs become too negative for exponentiation). See later sections for how we solved this issue.

Proposing a specific neural implementation for the generic belief propagation algorithm goes beyond the scope of this paper. However, we wanted to point out here that regardless of implementation details, corrections by previous messages (and thus "inhibitory loops") are required to map BP equations on a recurrent neural network.

2 BP in pairwise graphs

For the sake of simplicity, we now limit ourselves to pairwise graphs, i.e. to probability distributions that factorizes into functions of single and pairs of variables:

$$p(\mathbf{x}) = \prod_I f_{ij}(x_i, x_j) g(x_i) \tag{9}$$

In particular, any causal tree (as used in the paper) can be factorized this way.

In that case, we can dispense of the "factor node" completely. Indeed, consider a factor I connecting variable i and j (in a pairwise graph, a factor node can at most be connected to two variables). Then, $M_{I\to i}$ and $M_{i\to I}$ can we rewritten as M_{ji} and M_{ji} (see fig S1B,C). The BP equations simplify to:

$$B_i(x_i) = \sum_{j\in N_i} M_{ji}(x_i) \tag{10}$$

$$M'_{ji}(x_i) = W_{ji}\left(B_j(x_j) - M_{ij}(x_j)\right). \tag{11}$$

with

$$W_{ji}(B_j(x_j)) = \log\left(\sum_{x_i} f_{ij}(x_i, x_j)\exp\left(B_j(x_j)\right)\right) \tag{12}$$

As long as the order of updates is chosen properly, the BP algorithm does exact inference. In a tree-like graph, for example, this can be done by starting from the leaves (the sensory observation), climbing to the root (the higher level variables) and then descending back to the leaves. Only one pass of these updates equations (analogous to one pass of feed-forward processing followed by one pass of feedback processing) are sufficient. However, the algorithm also converges to the correct beliefs when update is synchronous (as in our simulations) or asynchronous and occurring in random order.

Since the variables in our examples are binary, we used log belief ratio rather than log beliefs, i.e. beliefs were represented by a single value $B_i = \log(\frac{b_i(1)}{b_i(0)})$ rather than two values $B_i(1) = \log(b_i(1))$ and $B_i(0) = \log(b_i(0))$. This we can do because $b_i(1) + b_i(0) = 1$, and thus $B_i = \log(\frac{b_i(1)}{1-b_i(1)})$. Similarly we used a single message value $M_{ij} = \log(\frac{M_{ij}(1)}{M_{ij}(0)})$. This change in variable gets rid of the normalization term, and thus of the numerical problem mentioned above. The resulting equations are provided in the main text.

3 Learning the causal links

To learn the causal relationships p^1_{ij} and p^0_{ij}, we used the EM algorithm, which (if the BP algorithm is exact) converges to the maximum-likelihood estimates (i.e. the parameters of the causal models maximizing the likelihood of the training examples). As its name indicates, this algorithm alternates two stages until convergence: An *expectation stage*, which finds the expected values of the hidden variables (i.e. the beliefs) given the current parameter estimates and the training examples; and a *maximization* stage, which updates the parameters to maximize the likelihood of the training examples given these expected values of the hidden states. To model the progressive learning of causal relationships based on training sequences, we used an on-line version of the EM algorithm, where the parameters p^1_{ij} and p^0_{ij} were re-estimated after each block of 50 new trials.

Let us consider two variables, x_i and x_j, where i presumably cause j (i.e. i is above j). We note E^t_i all the top-down evidence (i.e. excluding information from x_j) received by x_i in training trial t, and E^t_j all the bottom-up evidence (i.e. excluding information from x_i) received by x_j in training trial t. The expectation stage consists in computing the expected values for x^t_i given E^t_i in trial t, and the expected values for x^t_j given E^t_j in trial t, for the current sets of parameters p^k_{ij}. These can be obtained by running the BP (or LBP) algorithm and computing $\hat{x}^t_i = h(B_i - \alpha_d M_{ij})$ and $\hat{x}^t_j = h(B_j - \alpha_c M_{ji})$,

where B_i and M_{ij} correspond to the final messages and beliefs after completion of the algorithm (convergence for BP, 10 iterations for LBP). h is the logistic function $h(x) = (1 + e^{-x})^{-1}$. The expected value for all combinations of states are given by

$$a_{11}^t = \frac{\hat{x}_i^t p_{ij}^1 \hat{x}_j^t}{Z_t} \quad (13)$$

$$a_{10}^t = \frac{\hat{x}_i^t (1 - p_{ij}^1)(1 - \hat{x}_j^t)}{Z_t} \quad (14)$$

$$a_{01}^t = \frac{(1 - \hat{x}_i^t) p_{ij}^0 \hat{x}_j^t}{Z_t} \quad (15)$$

$$a_{00}^t = \frac{(1 - \hat{x}_i^t)(1 - p_{ij}^0)(1 - \hat{x}_j^t)}{Z_t}. \quad (16)$$

where Z_t is a normalization term such that $\sum_{kl} a_{kl}^t = 1$.

Once every 50 trials, the parameters p_{ij}^1 and p_{ij}^0 were replaced by their new updated value:

$$p_{ij}^{1*} = \frac{\sum_{t<T} a_{11}^t}{\sum_{t<T} a_{11}^t + a_{10}^t} \quad (17)$$

$$p_{ij}^{0*} = \frac{\sum_{t<T} a_{01}^t}{\sum_{t<T} a_{01}^t + a_{00}^t} \quad (18)$$

with $\sum_{t<T}$ representing the sum over all past training examples. The EM algorithm is often the "gold-standard" in machine learning (when inference is tractable). However, the multiple forms of normalization required are not biologically plausible. We also tested a more plausible algorithm where the parameters are updated by stochastic gradient descent. The modified learning rule becomes:

$$p_{ij}^{1*} = \frac{\sum_{t<T} \tilde{x}_i^t \tilde{x}_j^t}{\sum_{t<T} \tilde{x}_i^t} \quad (19)$$

$$p_{ij}^{0*} = \frac{\sum_{t<T} (1 - \tilde{x}_i^t) \tilde{x}_j^t}{\sum_{t<T} (1 - \tilde{x}_i^t)} \quad (20)$$

with $\tilde{x}_i = H(B_i - \alpha_d M_{ij})$ and $\tilde{x}_j = H(B_j - \alpha_c M_{ji})$. H is the heavyside function, i.e. $H(x) = 0$ in $x < 0$ and $H(x) = 1$ otherwise. This rule is analogous to a Hebbian learning rule. Results, shown on supplementary figure S3, are qualitatively similar to those reported in the main paper.

The training examples used for learning p_{23}^1 and p_{23}^0 in figure 3 and supplementary figure 3 were generated as follow. First the state x_1^t (the state of the "forest" node) was sampled from a binomial distribution with $p = 0.5$. Next, the state of the "tree" node x_2^t was sampled from a binomial distribution with probability $p_{12}^{1\text{true}} x_1^t + p_{12}^{0\text{true}} (1 - x_1^t)$. Next, the state of the "leaf" node x_3 was sampled from a binomial distribution with probability $p_{23}^{1\text{true}} x_2^t + p_{23}^{0\text{true}} (1 - x_2^t)$. Finally, the state of the "green" node x_5^t was sampled with probability $p_{35}^{1\text{true}} x_3^t + p_{35}^{0\text{true}} (1 - x_3^t)$. To compute beliefs, the BP and LBP algorithm were run while clamping message $M_1^t = 100 x_t^1 - 100(1 - x_t^1)$ and $M_5^t = 100 x_t^5 - 100(1 - x_t^5)$. This corresponds to assuming that "green" and "forest" are unambiguously observed (i.e. their beliefs are (almost) [0,1] or [1,0], corresponding to a perfect certainty that the corresponding variables are 1 or 0).