Prediction-Centric Learning of Independent Cascade Dynamics from Partial Observations – Supplementary Materials

Mateusz Wilinski [1] **Andrey Y. Lokhov** [1]

1. Details of SLICER implementation

1.1. Full derivation of the algorithm

For consistency, let us re-state the DMP equations for the IC model used in the Main Text, Eqs. (1)-(2):

$$p_i^c(t) = 1 - \big(1 - \bar{p}_i^c\big) \prod_{k \in \partial i} \Big(1 - \alpha_{ki} \cdot p_{k \to i}^c(t-1)\Big), \tag{S1}$$

$$p_{j \to i}^c(t) = 1 - \big(1 - \bar{p}_j^c\big) \prod_{k \in \partial j \setminus i} \Big(1 - \alpha_{kj} \cdot p_{k \to j}^c(t-1)\Big), \tag{S2}$$

which allow to compute $p_i^c(t)$ - the probability of node i to be active at time t, under the initial condition for the cascade c. As specified in the Main Text, assuming that each cascade has a single node source, we can write the objective function as:

$$\mathcal{O} = \sum_{s \in S} \sum_{i \in O} \sum_{\tau_i^s} m^{\tau_i^s} \log \mu_i^s(\tau_i^s), \tag{S3}$$

where $m^{\tau_i^s}$ is the number of times when node i was activated at time τ_i^s for all cascades in the data that start at node s, and $\mu_i^s(\tau_i^c)$ is the marginal probability that node i for an initial source s is activated precisely at time τ_i^s. These marginals can be approximately expressed through the DMP equations, which results in the following expression:

$$\mathcal{O} = \sum_{s \in S} \sum_{i \in O} \sum_{\tau_i^s} m^{\tau_i^s} \log \Big(p_i^s(\tau_i^s) \cdot \mathbb{1}_{(\tau_i^s < T)} - p_i^s(\tau_i^s - 1) \cdot \mathbb{1}_{(\tau_i^s > 0)} + \mathbb{1}_{(\tau_i^s = T)}\Big), \tag{S4}$$

where T is the maximum cascade length.

The constraints are given by DMP equations re-weighted by Lagrange multipliers:

$$\begin{aligned} \mathcal{C} = &\sum_{s \in S} \sum_{t=0}^{T-1} \sum_{i \in V} \lambda_i^s(t+1) \left(p_i^s(t+1) - 1 + \big(1 - \bar{p}_i^s\big) \prod_{k \in \partial i} \Big(1 - \alpha_{ki} \cdot p_{k \to i}^s(t)\Big) \right) \\ &+ \sum_{s \in S} \sum_{t=0}^{T-1} \sum_{(i,j) \in E} \lambda_{i \to j}^s(t+1) \left(p_{i \to j}^s(t+1) - 1 + \big(1 - \bar{p}_i^s\big) \prod_{k \in \partial i \setminus j} \Big(1 - \alpha_{ki} \cdot p_{k \to i}^s(t)\Big) \right), \end{aligned} \tag{S5}$$

where $\lambda_i^s(t)$ and $\lambda_{i \to j}^s(t)$ are Lagrange multipliers corresponding respectively to marginals from (S1) and messages from (S2).

[1]Theoretical Division Los Alamos National Laboratory, Los Alamos, USA. Correspondence to: Mateusz Wilinski <mateusz@lanl.gov>, Andrey Y. Lokhov <lokhov@lanl.gov>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

The resulting expressions for all the quantities can be found by differentiating the Lagrangian. Differentiation over marginals yields expressions for $\lambda_i^s(t)$ multipliers for all sources s and times t:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial p_i^s(t)} = \lambda_i^s(t) &+ \sum_{\tau_i^s} \frac{m^{\tau_i^s} \cdot \mathbb{1}_{(t=\tau_i^s)} \cdot \mathbb{1}_{(\tau_i^s<T)}}{p_i^s(\tau_i^s) - p_i^s(\tau_i^s-1) \cdot \mathbb{1}_{(\tau_i^s>0)}} \\ &+ \sum_{\tau_i^s} \frac{m^{\tau_i^s} \cdot \mathbb{1}_{(t=\tau_i^s-1)} \cdot \mathbb{1}_{(\tau_i^s>0)}}{p_i^s(\tau_i^s-1) - p_i^s(\tau_i^s) \cdot \mathbb{1}_{(\tau_i^s<T)} - \mathbb{1}_{(\tau_i^s=T)}}.\end{aligned} \tag{S6}$$

One direct consequence of this is that $\lambda_i^s(t) = 0 \, \forall_{t \notin \{\tau_i^s, \tau_i^s - 1\}}$. The $\lambda_{i\to j}^s(t)$ multipliers can be obtained from the derivatives over messages. For $t < T$ we get

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial p_{i\to j}^s(t)} = \lambda_{i\to j}^s(t) &- \lambda_j^s(t+1)\, \alpha_{ij} \left(1-\bar{p}_j^s\right) \prod_{m\in\partial j\setminus i} \left(1-\alpha_{mj}\cdot p_{m\to j}^s(t)\right) \\ &- \sum_{k\in\partial j\setminus i} \lambda_{j\to k}^s(t+1)\, \alpha_{ij} \left(1-\bar{p}_j^s\right) \prod_{m\in\partial j\setminus\{i,k\}} \left(1-\alpha_{mj}\cdot p_{m\to j}^s(t)\right).\end{aligned} \tag{S7}$$

For $t = T$, the expression simplifies to

$$\frac{\partial \mathcal{L}}{\partial p_{i\to j}^s(T)} = \lambda_{i\to j}^s(T), \tag{S8}$$

which allows to calculate all $\lambda_{i\to j}^s(t)$ in an inductive manner, starting from T and using $\lambda_{i\to j}^s(t+1)$ to compute multipliers for $t < T$. Finally, derivatives over parameters α_{ij} read

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \alpha_{ij}} = &- \sum_{s\in S}\sum_{t=0}^{T-1} \lambda_i^s(t+1)\, p_{j\to i}^s(t) \left(1-\bar{p}_i^s\right) \prod_{m\in\partial i\setminus j} \left(1-\alpha_{mi}\cdot p_{m\to i}^s(t)\right) \\ &- \sum_{s\in S}\sum_{t=0}^{T-1} \lambda_j^s(t+1)\, p_{i\to j}^s(t) \left(1-\bar{p}_j^s\right) \prod_{m\in\partial j\setminus i} \left(1-\alpha_{mj}\cdot p_{m\to j}^s(t)\right) \\ &- \sum_{s\in S}\sum_{t=0}^{T-1}\sum_{k\in\partial i\setminus j} \lambda_{i\to k}^s(t+1)\, p_{j\to i}^s(t) \left(1-\bar{p}_i^s\right) \prod_{m\in\partial i\setminus\{j,k\}} \left(1-\alpha_{mi}\cdot p_{m\to i}^s(t)\right) \\ &- \sum_{s\in S}\sum_{t=0}^{T-1}\sum_{k\in\partial j\setminus i} \lambda_{j\to k}^s(t+1)\, p_{i\to j}^s(t) \left(1-\bar{p}_j^s\right) \prod_{m\in\partial j\setminus\{i,k\}} \left(1-\alpha_{mj}\cdot p_{m\to j}^s(t)\right),\end{aligned} \tag{S9}$$

which can further be simplified for $\alpha_{ij} \neq 0$:

$$\frac{\partial \mathcal{L}}{\partial \alpha_{ij}} = \frac{-1}{\alpha_{ij}} \sum_{s\in S}\sum_{t=0}^{T-1} \left(\lambda_{i\to j}^s(t)\, p_{i\to j}^s(t) + \lambda_{j\to i}^s(t)\, p_{j\to i}^s(t)\right). \tag{S10}$$

Now, we can use gradient components (S27) to update parameters α_{ij}:

$$\alpha_{ij} \longleftarrow \alpha_{ij} + \varepsilon \cdot \frac{\partial \mathcal{L}}{\partial \alpha_{ij}}, \tag{S11}$$

where ε is a learning rate. The way we specify this learning rate is described in section 1.2.

1.2. Choice of the learning rate

All of the numerical results presented in the paper were obtained with the learning rate $\varepsilon = c\frac{N}{MT}$ in equation (S11), where N is the number of nodes, M is the number of cascades, T is the length of cascades, and c is a small constant (the same for all networks) ensuring that the step is not too large (in our experiments, we used $c = 1/80$). This normalisation of the gradient step is convenient because it results in very similar number of steps until global convergence across different network sizes for similar level of available information. This effect is demonstrated in Fig. S1 in the case of random three regular graph with varying size and accordingly re-scaled number of cascades. As a consequence of this observation, it is easy to estimate the running time of the algorithm based on the running time of a single iteration. In the next section, we show the empirical scalability per iteration step of the proposed algorithm as a function of problem parameters.

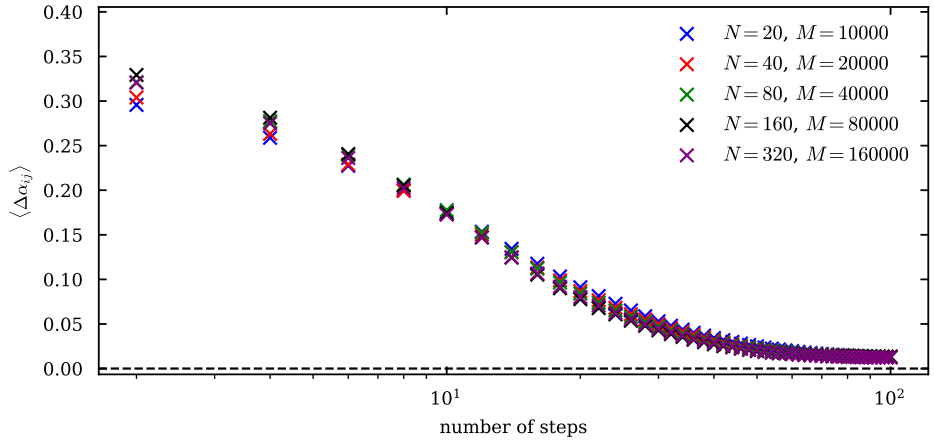

Figure S1. Convergence of the algorithm as a function of number of gradient descent steps, measured by the change of α_{ij} after each consecutive step. The simulations were done with cascades of length $T = 10$. All the points were obtained using the learning rate $\varepsilon = c\frac{N}{MT}$ in the equation (S11).

1.3. Empirical algorithmic complexity

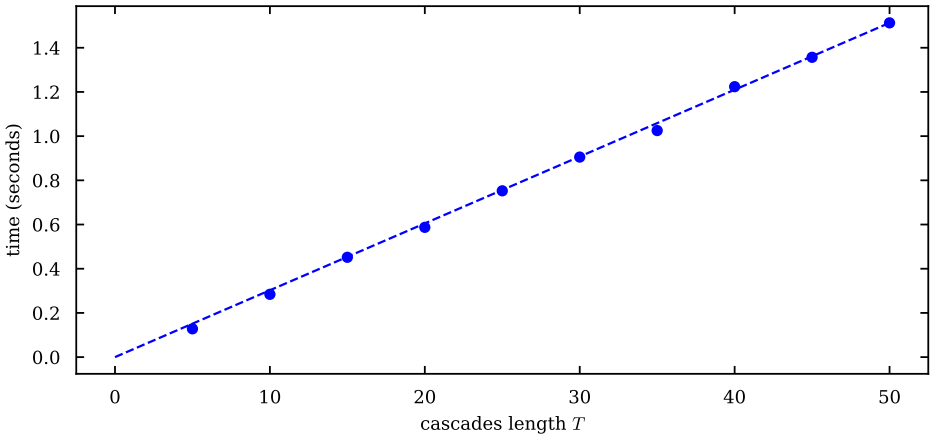

Figure S2. Averaged computational times for a single step of optimisation, as a function of cascade length T. Simulations were made using $M = 100$ cascades on a random three regular graph with $N = 100$ nodes. Each point represents an average over five different realisations of the network. The reference dashed line is the best linear fit with a zero intercept.

As discussed in the Main Text, the overall computational complexity of SLICER is $O(|E|T|S|)$. In this section, we check the empirical scalability of the algorithm. Fig. S2 shows the linearly growing computational time required to perform one step of the optimisation process as a function of the cascade length T. A more interesting dependence on the number of cascades is shown in Fig. S3. Linear until the number of cascades reaches the order of the network size, the optimization time starts to plateau for larger M once the number of classes saturates to $|S| = N$ (we only use single node seeds in this experiment), which is consistent with the complexity analysis above. Note that for stochastic initial conditions $|S| = 1$. Finally, the linear scaling for different network sizes is presented in Fig. S4.

1.4. Code and implementation efficiency

The code used to obtain all the results in the paper is available at: `https://github.com/mateuszwilinski/dynamic-message-passing`. We used a straightforward implementation of the proposed algorithm, where the DMP

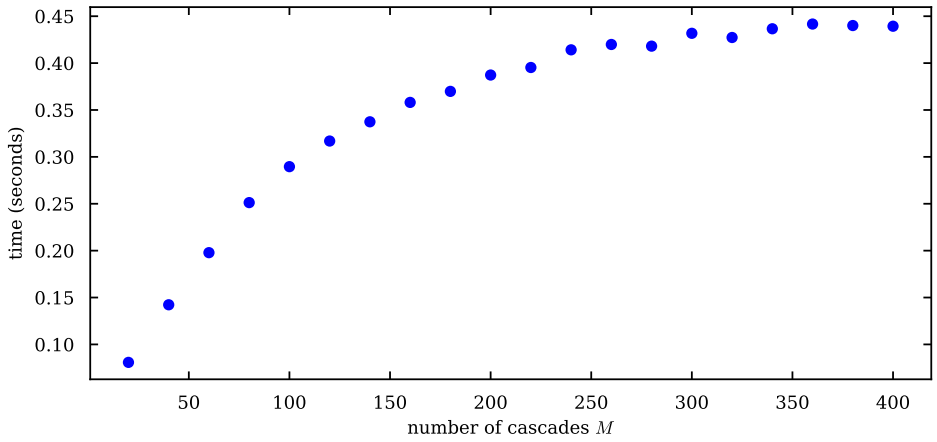

Figure S3. Averaged computational times for a single step of optimisation, as a function of the number of available cascades M. Simulations were made using cascades of $T = 10$ length on a random three regular graph with $N = 100$ nodes. Each point represents an average over five different realisations of the network.

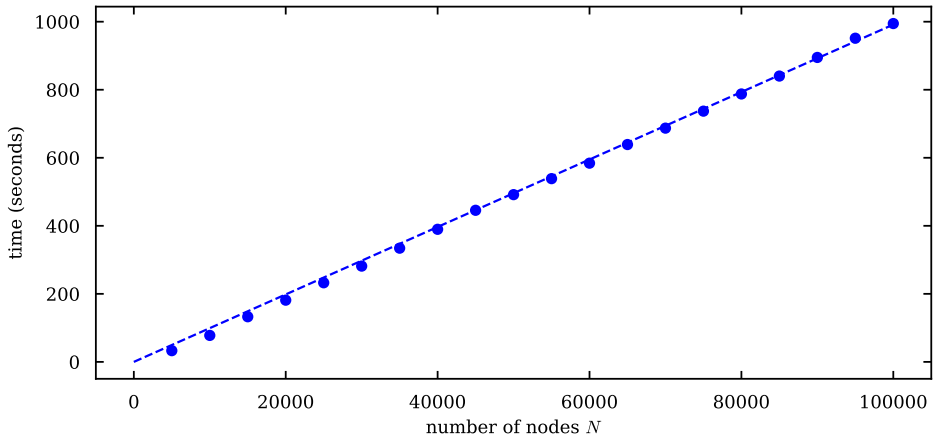

Figure S4. Averaged computational time for a single step of optimisation, as a function of the network size N. Simulations were made using $M = 100$ cascades of $T = 10$ length on a random three regular graphs of different sizes. Each point represents an average over five different realisations of the network. The reference dashed line is the best linear fit with a zero intercept.

equations are run for all nodes and edges in the graph. This implementation choice was made so that it is applicable to general initial conditions that include arbitrary initial probability distributions factorized over nodes of the graph (Lokhov & Saad, 2019). However, for the single-source cascades considered in this paper for simplicity, it is clear that on very large networks one could take advantage of the finite T and restrict the dynamics both in the primal and in the dual space to the part of the network that is reachable by the dynamics. Due to the linear scaling with the system size our learning scheme can be run on networks with the size on the order of millions of nodes. At these sizes the implementation could be further improved by a more efficient memory usage. Finally, notice that calculations for different cascades can be paralleled, where additionally an asynchronous stochastic version of the gradient descent can be used.

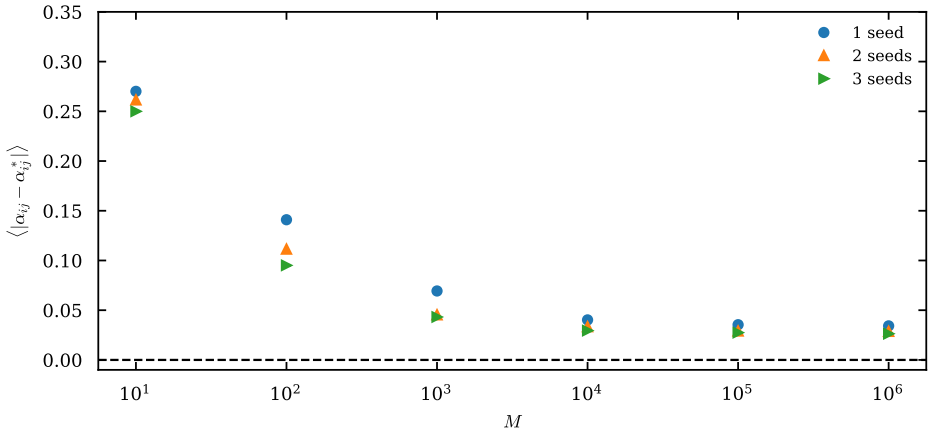

Figure S5. Results for Erdös-Rényi network with $N = 100$ nodes, average degree of 3, cascades length $T = 10$, uniformly distributed parameters α_{ij} and different number of initially activated nodes. Seed size has little impact on reconstruction quality if single-source statistics has big network coverage.

2. Further experiments on the experimental setting

2.1. Effect of initial number of activated nodes

In the paper, we assume that cascades are random, and each one can start from any randomly-selected node. To address the influence of initial conditions, we ran an experiment that shows reconstruction error as a function of the seed set size. Results in Fig. S5, on a heterogeneous Erdös-Rényi network, show that increase in the seed size has a minor impact on the error if all nodes are activated a sufficient number of times, even if this happens in small-outbreak cascades with different sources.

2.2. Effects of network topology and placement of the missing nodes in the network

In the paper, we deliberately presented synthetic instances mostly with regular degree (grids and random graphs) to eliminate potential heterogeneity on the distribution of hidden nodes. Nevertheless, a question on the influence of hidden node distribution is still interesting. We ran additional experiments on a heterogeneous Erdös-Rényi network, comparing results for hidden node distributions targeting primarily high- or low-degree nodes with the random case, and otherwise in the same setting as throughout the paper. The results in Table S1 show better accuracy for hidden high-degree nodes due to dynamics going through them more frequently and hence accumulating more statics.

hidden location	high-degree	random	low-degree
average ℓ_1 error	0.032	0.039	0.047

Table 1. Average reconstruction error for the parameters as a function of distribution of hidden nodes location on an Erdös-Rényi network with $N = 100$, $T = 10$, $M = 10^5$, and $\xi = 15\%$.

3. DMPrec derivation for the IC model

The idea behind DMPrec is to compute the derivative over the DMP free energy of the system. The part of the free energy related to individual node i can be written as:

$$f^i_{\text{DMP}} = -\sum_c \log \mu^c_i(\tau^c_i), \tag{S12}$$

where marginal probability $\mu^c_i(\tau^c_i)$ of node i being activated at time t during cascade c can be computed using marginals from DMP equations (S1):

$$\mu^c_i(t) = p^c_i(t)\mathbb{1}_{(t<T)} - p^c_i(t-1)\mathbb{1}_{(t>0)} + \mathbb{1}_{(t=T)}. \tag{S13}$$

The derivative of the DMP free energy over the spreading probabilities is as follows:

$$\frac{\partial f^i_{\text{DMP}}}{\partial \alpha_{rs}} = -\sum_c \frac{1}{\mu^c_i(\tau^c_i)} \frac{\partial \mu^c_i(\tau^c_i)}{\partial \alpha_{rs}}. \tag{S14}$$

For cascades originating from a single source, we can further write:

$$\frac{\partial f^i_{\text{DMP}}}{\partial \alpha_{rs}} = -\sum_s \sum_{\tau^s_i} \frac{m^{\tau^s_i}}{\mu^s_i(\tau^s_i)} \frac{\partial \mu^s_i(\tau^s_i)}{\partial \alpha_{rs}}. \tag{S15}$$

Let us now denote:

$$\frac{\partial p^c_{k\to i}(t)}{\partial \alpha_{rs}} = \phi^{rs}_{k\to i}(t). \tag{S16}$$

Note that $\phi^{rs}_{k\to i}(0) = 0$. We can use that and compute values for all other times in an iterative way, using:

$$\phi^{rs}_{k\to i}(t) = \big(1-\bar{p}^c_k\big) \sum_{l\in\partial k\setminus i} \Big(\alpha_{lk}\cdot \phi^{rs}_{l\to k}(t-1) + p^c_{l\to k}(t-1)\mathbb{1}_{l=r,k=s}\Big) \prod_{j\in\partial k\setminus\{i,l\}} \Big(1-\alpha_{jk}\cdot p^c_{j\to k}(t-1)\Big), \tag{S17}$$

which comes from differentiating messages (S2). Finally, we are able to compute the derivative in (S15) using:

$$\begin{aligned}\frac{\partial \mu^c_k(t)}{\partial \alpha_{rs}} = &\big(1-\bar{p}^c_k\big) \sum_{l\in\partial k} \Big(\alpha_{lk}\cdot \phi^{rs}_{l\to k}(t-1) + p^c_{l\to k}(t-1)\mathbb{1}_{l=r,k=s}\Big) \prod_{j\in\partial k\setminus l} \Big(1-\alpha_{jk}\cdot p^c_{j\to k}(t-1)\Big)\mathbb{1}_{(t<T)} \\ &- \big(1-\bar{p}^c_k\big) \sum_{l\in\partial k} \Big(\alpha_{lk}\cdot \phi^{rs}_{l\to k}(t-2) + p^c_{l\to k}(t-2)\mathbb{1}_{l=r,k=s}\Big) \prod_{j\in\partial k\setminus l} \Big(1-\alpha_{jk}\cdot p^c_{j\to k}(t-2)\Big)\mathbb{1}_{(t>0)}.\end{aligned} \tag{S18}$$

The derivative (S15) can then be used instead of $\frac{\partial \mathcal{L}}{\partial \alpha_{ij}}$ equation (S11) and iterated until convergence in the same manner as we did in the SLICER's framework.

4. Maximum Likelihood approach for the full observation case

For completeness, we explain the maximum likelihood implementation that we used as a baseline in the Main Text for comparisons in the case of full observability. The probability of observed data $\Sigma = \bigcup_{c=1}^M \Sigma^c$, where $\Sigma^c \equiv \{\tau^c_i\}_{i\in V}$, given a set of spreading parameters $\Omega \equiv \{\alpha_{ij}\}_{(i,j)\in E}$, can be computed using the following form factorised over individual nodes:

$$P(\Sigma|\Omega) = \prod_{c\in C}\prod_{i\in V} P_i(\tau^c_i|\Sigma^c,\Omega), \tag{S19}$$

with

$$P_i(\tau^c_i|\Sigma^c,\Omega) = \left(\prod_{k\in\partial i}(1-\alpha_{ki}\mathbf{1}_{\tau^c_k\le\tau^c_i-2})\right)\cdot\left(1-\prod_{k\in\partial i}(1-\alpha_{ki}\mathbf{1}_{\tau^c_k=\tau^c_i-1})\mathbf{1}_{\tau^c_i<T}\right), \tag{S20}$$

where ∂i denotes the set of neighbors of node i. Note that equation (S20) is only valid for $\tau^c_i > 0$, otherwise the probability is equal to 1. This allows to get an estimate of the spreading parameters $\Omega^* \equiv \{\alpha^*_{ij}\}_{(i,j)\in E}$ by solving a convex optimisation problem

$$\Omega^* = \arg\min_{\Omega}\left(-\log P(\Sigma|\Omega)\right), \tag{S21}$$

under constraints:

$$0 \le \alpha_{ij} \le 1 \quad \forall_{(i,j)\in E}. \tag{S22}$$

The results presented in the Main Text were obtained with the Ipopt solver (Wächter & Biegler, 2006), using the implementation of (S21)-(S22) within the JuMP framework for non-linear optimization (Dunning et al., 2017).

5. Details of the multi-replica approach

5.1. Learning framework for the replicas

Here, we provide a detailed algorithm that we used for constructing multi-replica effective mpdels in the Main Text. In order to account for the error in the DMP equations that result from the presence of loops, instead of having one network with a set of parameters $\{\alpha_{ij}\}$ we consider a set R of structurally identical networks with a unique set of parameters $\{\alpha^r_{ij}\}$ each. In such setting marginal probability $\mu^s_i(t)$ of activating node i at time t under cascade starting from node s is equal to an average over marginals in each replica. This leads to a following objective function presented in the Main Text and re-stated here for consistency:

$$\mathcal{O}^{\text{mixture}} = \sum_{s\in S}\sum_{i\in O}\sum_{\tau^s_i} m^{\tau^s_i}\log\left(\frac{1}{|R|}\sum_{r\in R}\mu^s_{i,r}(\tau^s_i)\right). \tag{S23}$$

This objective can be rewritten using approximate marginals obtained separately from DMP equations for each replica:

$$\mathcal{O}^{\text{mixture}} = \sum_{s\in S}\sum_{i\in O}\sum_{\tau^s_i} m^{\tau^s_i}\log\left(\frac{1}{|R|}\sum_{r\in R}\left(p^{r,s}_i(\tau^s_i)\cdot\mathbb{1}_{(\tau^s_i<T)} - p^{r,s}_i(\tau^s_i-1)\cdot\mathbb{1}_{(\tau^s_i>0)} + \mathbb{1}_{(\tau^s_i=T)}\right)\right), \tag{S24}$$

where $p^{r,s}_i(t)$ is the marginal probability of node i being activated at time t or earlier by a cascade originated from s, under replica r. Following the procedure described for SLICER in 1.1 we compute a Lagrangian for the multi-replica objective function. We will require separate sets of Lagrange multipliers for each replica. We define $\lambda^{r,s}_i(t)$ and $\lambda^{r,s}_{i\to j}(t)$ as the Lagrange multipliers for marginal probability $p^{r,s}_i(t)$ and message $p^{r,s}_{i\to j}(t)$ equations respectively. The update equations for the multipliers can be computed using the derivative of the Lagrangian $\mathcal{L}$ over the DMP marginals:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial p^{r,s}_i(t)} = \lambda^{r,s}_i(t) &+ \sum_{\tau^s_i}\frac{m^{\tau^s_i}\cdot\mathbb{1}_{(t=\tau^s_i)}\cdot\mathbb{1}_{(\tau^s_i<T)}}{\sum_{r\in R}\left(p^{r,s}_i(\tau^s_i) - p^{r,s}_i(\tau^s_i-1)\cdot\mathbb{1}_{(\tau^s_i>0)}\right)}\\ &+ \sum_{\tau^s_i}\frac{m^{\tau^s_i}\cdot\mathbb{1}_{(t=\tau^s_i-1)}\cdot\mathbb{1}_{(\tau^s_i>0)}}{\sum_{r\in R}\left(p^{r,s}_i(\tau^s_i-1) - p^{r,s}_i(\tau^s_i)\cdot\mathbb{1}_{(\tau^s_i<T)} - \mathbb{1}_{(\tau^s_i=T)}\right)}.\end{aligned} \tag{S25}$$

The message-multipliers can be obtained using the derivatives over DMP messages and previously obtained marginal-multipliers:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial p^{r,s}_{i\to j}(t)} = \lambda^{r,s}_{i\to j}(t) &- \lambda^{r,s}_j(t+1)\,\alpha^r_{ij}\left(1-\bar{p}^s_j\right)\prod_{m\in\partial j\setminus i}\left(1-\alpha^r_{mj}\cdot p^{r,s}_{m\to j}(t)\right)\\ &- \sum_{k\in\partial j\setminus i}\lambda^{r,s}_{j\to k}(t+1)\,\alpha^r_{ij}\left(1-\bar{p}^s_j\right)\prod_{m\in\partial j\setminus\{i,k\}}\left(1-\alpha^r_{mj}\cdot p^{r,s}_{m\to j}(t)\right).\end{aligned} \tag{S26}$$

Finally one can use the Lagrange multipliers to compute the derivatives over the spreading parameters:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\alpha^r_{ij}} = &-\sum_{s\in S}\sum_{t=0}^{T-1}\lambda^{r,s}_j(t+1)\,p^{r,s}_{i\to j}(t)\left(1-\bar{p}^s_j\right)\prod_{m\in\partial j\setminus i}\left(1-\alpha^r_{mj}\cdot p^{r,s}_{m\to j}(t)\right)\\ &-\sum_{s\in S}\sum_{t=0}^{T-1}\sum_{k\in\partial j\setminus i}\lambda^{r,s}_{j\to k}(t+1)\,p^{r,s}_{i\to j}(t)\left(1-\bar{p}^s_j\right)\prod_{m\in\partial j\setminus\{i,k\}}\left(1-\alpha^r_{mj}\cdot p^{r,s}_{m\to j}(t)\right).\end{aligned} \tag{S27}$$

In case of non-zero parameters α^r_{ij}, the above equation can be significantly simplified:

$$\frac{\partial\mathcal{L}}{\partial\alpha^r_{ij}} = -\frac{1}{\alpha^r_{ij}}\sum_{s\in S}\sum_{t=0}^{T-1}\left(\lambda^{r,s}_{i\to j}(t)\,p^{r,s}_{i\to j}(t) + \lambda^{r,s}_{j\to i}(t)\,p^{r,s}_{j\to i}(t)\right). \tag{S28}$$

As before, the above derivatives serve as a gradient to update parameters α^r_{ij}:

$$\alpha^r_{ij} \longleftarrow \alpha^r_{ij} + \varepsilon\cdot\frac{\partial\mathcal{L}}{\partial\alpha^r_{ij}}. \tag{S29}$$

Notice that for a single replica, this procedure reduces to the SLICER algorithm described in section 1.1.

5.2. Remarks on the iterative improvement of the effective model

We present empirical tests showing that due to the non-convexity of the problem, random initializations lead to different local solutions with a similar prediction accuracy. This phenomenon is illustrated in Fig. S6, where we test 10 different initial conditions (here, for a single-replica case).

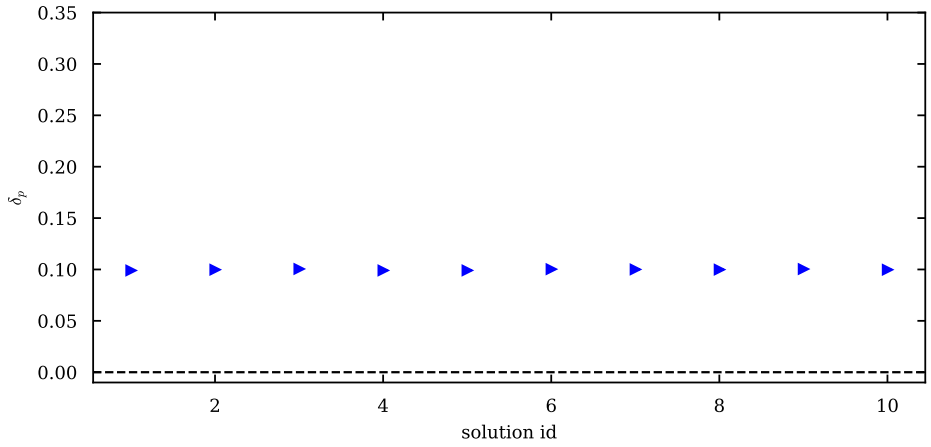

Figure S6. Marginal prediction error for different random initialisation of the single-replica effective model. The results were obtained for a square lattice with $N = 100$ nodes and $M = 10^6$ cascades of length $T = 20$.

A scatter plot of two solutions obtained with different random initializations is shown in Fig. S7(a), where we see that although the reconstructed parameters are different, they are strongly aligned with each other. This observation motivates the multi-replica approach, which suggests that instead of spending the computational budget by warm-starting SLICER multiple times, it is beneficial to invest computing time into improving a given single-replica solution. Fig. 6 of the Main Text suggested that for a fixed number of replicas, an iterative improvement of the solution starting with an initialization given by a slightly perturbed (to break the symmetry) solution obtained with for an effective model with a smaller number of replicas is more advantageous compared to the search of an optimal multi-replica solution using random initialization. Indeed, this procedure guarantees the improvement of the solution as long as the multi-replica objective is maximized, while the search of the parameters of the multi-replica model from random initialization typically leads to a lower-quality solution as the gradient descent is likely stuck at a local optimum (this was the case in all test experiments we ran). Fig. S7(b) shows an example of two replica solution (one of its layers) against a single replica solution that was used as initialization. We see that the parameters diverge, trying to compensate errors arising from the loops in the graph and better match the empirical marginal probability distributions.

References

Dunning, I., Huchette, J., and Lubin, M. Jump: A modeling language for mathematical optimization. *SIAM review*, 59(2): 295–320, 2017.

Lokhov, A. Y. and Saad, D. Scalable influence estimation without sampling. *arXiv preprint arXiv:1912.12749*, 2019.

Wächter, A. and Biegler, L. T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

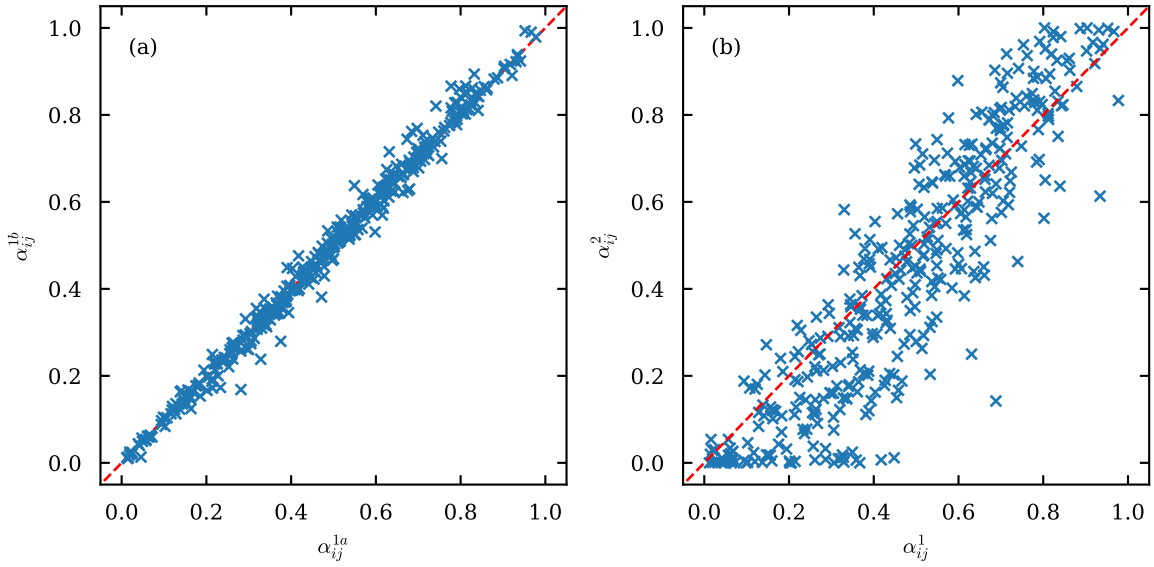

Figure S7. Scatter plots of parameters α_{ij} obtained for different replicas. The results were obtained for a square lattice with $N = 100$ nodes and $M = 10^6$ cascades of length $T = 20$. (a) Comparison of parameters inferred for a single-replica model with different SLICER initializations. (b) Comparison of one of the layers of the 2-replica solution and the single-replica model used as an initialization to the 2-replica SLICER algorithm.