Controlling Graph Dynamics with Reinforcement Learning and Graph Neural Networks : Supplementary Material

A. Motivating example details

We provide the details for the example from Section 2 (Figure 3). Recall that Our goal is to minimize the number of infected nodes in a social interactions graph. A natural algorithmic choice would be to act upon nodes that are most likely infected. The following example shows why this approach is suboptimal.

We form a time-varying graph from a list of interactions between nodes at various times. If u, v interacted at time t then the edge $e = (u, v)$ exists at time t. Each interaction is characterized by a transmission probability $p_e(t)$. If a node was infected at time t and its neighbor was healthy, then the healthy node is infected with probability $p_e(t)$. Assume that we can test a single node at odd timesteps. If the node is identified as infected, it is sent to quarantine and cannot further interact with other nodes. Otherwise, we do not perturb the dynamics and it may interact freely with its neighbors.

Consider the ”two stars” network in Figure 3. The left hub (node v_1) has m_1 neighbors, and m_2 nodes are attached to the right hub v_2, with $m2 \gg m1$. At $t = 0$, only the edge $e = (v_1, v_2)$ is present with $p_e(t = 0) = p$. Then, for all $t \geq 1$, all edges depicted in Figure 3 exist with $p_e(t) = 1$. Assume that this information is known to the agent, and that at $t = 1$ we suspect that node v_1 was infected at $t = 0$.

If v_1 would turn out to be healthy, than any test would result in a negative result and would not affect the dynamics. Hence, in the following derivation we condition on node v_1 being infected at $t = 0$. Note that we can not quarantine prematurely v_1 unless detected as positive. In this case, we clearly should test either v_1 or v_2. We can compute the expected cost of each option exactly. **Alternative I:** Test v_2. With probability p, v_2 becomes infected at $t = 1$, and we block the epidemic from spreading. However, we forfeit protecting v_1 neighbors, as all of them will be infected in the next step. With probability $1-p$ the test is negative, and we fail to affect the dynamics. At $t = 2$ node v_2 will get infected and at $t = 3$ all of v_2's neighbors become infected too, ending up with a total of $(m_2 + 1)$ infections. The expected cost in choosing to test v_2 is $(1 - p) \cdot m_2 + m_1$. **Alternative II:** Test v_1. We block the spread to v_1's neighbors, but sacrifice all m_2 neighbors of v_2 with probability p. The expected cost in choosing v_2 is $p \cdot m_2$. The decision would therefore be to test for v_2 if $2p \geq 1 + m_1/m_2$.

This example illustrates that an optimal policy must balance two factors: *the probability that the dynamics is affected* - that a test action on v_2 yields a "positive", measured by p, and the future consequences of our action - the *strategic importance* of selecting v_1 *vs.* v_2, expressed by the ratio m_1/m_2. A policy targeting likely-infected nodes will always pick node v_1, but since it only focuses on the first term and ignores the second term, it is clearly suboptimal.

B. Problem Formulation - Additional Details

In this section we fill in on the details of our setup.

B.1. Epidemic test prioritization

The SEIR model dynamics (Lopez & Rodo, 2020). Every node (person) can be in one of the following states: *susceptible* – a healthy, yet uninfected person (S state), *exposed/latent* – infected but cannot infect others (L state), *infectious* – may infect other nodes (I state), or *removed* – self-quarantined and isolated from the graph (R state). Formally, let $\mathcal{I}(t) \subset V$ be the set of infectious nodes at time t, and similarly $\mathcal{L}(t)$, $\mathcal{R}(t)$ and $\mathcal{S}(t)$ be the sets of latent(exposed), removed and susceptible (healthy) nodes.

A healthy node can become infected by interacting with its neighbors. Each active edge at time t, $e \in \mathcal{E}(t)$, carries a transmission probability $p_e(t)$. Denote the set of impinging edges on node v with an infectious counterpart at time t by

$E_v(t)$. Formally,

$$E_v(t) = \{e \in \mathcal{E}(t) | e = (v, u), SV_u(t-1) = I\}.$$

The probability of a healthy node to remain healthy at time t is

$$\prod_{e \in E_v(t)} (1 - p_e(t))$$

otherwise it becomes infected, but still in a latent state. Denote the time of infection of node v as T_v. A node in a latent state will stay in this state at time t if $t < T_v + D_v$, where D_v is a random variable representing the latency period length, otherwise its state changes to infectious. The testing intervention changes the state of a node. If infected or exposed, its state is set to R, otherwise it remains as it is. In principle, a node is state R can be restored to the network after quarantining, though in our setup the quarantining period is larger than the episode duration and therefore once a node is in R it remains detached from the network for the rest of the simulation.

Optimization goal, action space. The objective is to minimize the spread of the epidemic, namely, minimize the number of infected people (in either L, R or I states),

$$\min \sum_{t,v} \gamma^t \left\| \mathcal{L}(t) \cup \mathcal{R}(t) \right\|,$$

where $\gamma \in (0, 1]$ is a discount factor representing the relative importance of the future compared to the present. We used $\gamma = 0.99$ throughout the paper.

The action space consists of all possible selections of a subset $a(t)$ of k nodes $a(t) \subset V$. Even for a moderate graph, with $\sim 100 - 1000$ and small k the action space $\binom{|\mathcal{V}|}{k}$ is huge.

B.2. Dynamic influence maximization

Model Dynamics. Each node is either *Influenced*, denoted by I or *Susceptible* (S). At each time the agent selects a seed set $a(t)$ of k nodes, and attempt to influence them to its cause. This succeeds with probability q independently for every $v \in a(t)$. Influenced nodes then propagate this cause, following a dynamic generalization of two canonical models: Linear Threshold (LT) and Independent Cascades (IC).

In a Linear Threshold dynamic model, each node v is associated with a threshold w_v, and each edge e carries an impact weight of q_e. The "peer pressure" $z_v(t)$ on a node is the total weight of active edges in the last T_{peer} steps connecting influenced neighboring nodes and node v.

$$z_v(t) = \sum_{e \in E_v(t)} q_e, \quad E_v(t) = \{(u, v) | (u, v) \in \mathcal{E}(t'), t - t' < T_{peer}, ST_u(t) = I\}$$

If the "peer pressure" on node v exceeds w_v, node v state is changed to *Influenced*.

In an Independent Cascades model, if u is Influenced and $(u, v) \in \mathcal{E}_t$, then u attempts to influence v. We explored two variations, IC(constant) and IC(geometric). In IC(constant), the success probability of each attempt is fixed as some p, while in IC(geometric), the success probability decays with the number of influence attempts $m_{(u,v)}$, so the success probability is $p^{m_{(u,v)}}$. This mimics the reduced effect of presenting the same information multiple times.

C. Approach Discussion

In this section we further motivate our design.

Policy gradients. An action-value approach like Q-learning implies that an approximate value is assigned to every possible action. The action space of choosing a subset of k nodes out of n nodes is prohibitively too large even for small n and k. Instead, we use a policy-gradient algorithm and model the problem as a ranking problem.

Many on-policy gradient algorithms use entropy to define a trust region. Computing the entropy requires summing $\binom{|\mathcal{V}|}{k}$ terms at each step, and it is computationally expensive. A more scalable solution is the unbiased entropy estimator of (Zhang et al., 2018b), but the variance of that estimator is high. As an alternative, PPO trust region is not based on an explicit evaluation of the entropy, and performed better in our experiments. We also evaluated A2C, which did not perform as well as PPO in our experiments.

Critic module. PPO, as an actor-critic algorithm, requires a critic module to estimate the value function in a given state. We construct the critic using an architecture similar to the ranking module. We apply an element-wise max operation on the rows (representing the nodes) of the score module F's input (Figure 5). This reduces F's input to a single row of features, and the output is then a scalar rather than a vector. Importantly, the critic is parametrized by a different set of weights than the ranking module (actor).

Normalization in scale-free networks. Recurrent neural networks are well-known to suffer from the problem of exploding or vanishing gradients. This problem is exacerbated in a RNN-GNN framweork. A node in Graph Neural Networks framework receives updates from a large number of neighbors and its internal state may increases in magnitude. The next time that the GNN module is applied (e.g., at the next RL step), the node's updates its neighbors, and its growing internal state updates and increases the magnitude of the internal state of its neighbors. This leads to a positive-feedback loop that causes the internal state representation to diverge. This problem is particularly severe if the underlying graph contains hubs (highly connected nodes). Scale-free networks contain with high probability "hub" nodes that have high-degree, namely $O(n)$ neighbors. The presence of these hubs further aggravates this problem. Since RL algorithms may be applied for arbitrary long periods, the internal state may grow unbounded unless corrected.

One approach to alleviate this problem is by including an RNN like a GRU module, where the hidden state values pass through a sigmoid layer. As the magnitude of the input grows, the gradient become smaller and training slows down. Alternatively, This problem can be solved by directly normalizing each node hidden state. We have experimented with various normalization methods, and found that L_2 normalization worked best, as shown in the next section.

Transition probabilities. In the case of the COVID-19 pandemic, the transition probabilities can be estimated using the interaction properties, such as duration and inter-personal distance, using known epidemiological models. This was done by the government agency which provided our contact tracing network (see below). Alternatively, one can learn the transmission probability as a regression problem from known interactions (e.g, using data from post-infection questioning). Finally, if this information is not accessible, it is possible to omit the epidemic model *E* from the proposed framework and use only the feature vector created by the information module *I*.

In a scale free network, there exists hubs with $O(n)$ neighbors. As a simple case, consider a star graph with a large number of nodes. In a GNN framework, it receives updates from a large number of neighbors, and its internal state increases in magnitude. In the next application of the GNN module, e.g., in the next RL step, its growing internal state will induce an increase in the magnitude of its neighbor's internal state, resulting in a positive feedback loop that will blow the internal state representation. Fundamentally, an RL algorithm may be applied for arbitrary long episodes, which will allow the internal state to grow unbounded.

D. Additional Experimental details

In this appendix we expand of various experimental aspects. We first elaborate on the different baselines.

D.1. Synthetic datasets

We study three types of networks which differ by their connectivity patterns.

(1) Community-based networks have nodes clustered into densely-connected communities, with sparse connections across communities. We use the *Stochastic Block Model* (SBM, (Abbe, 2017)), for 2 and 3 communities. The Stochastic Block Model (SBM) is defined by (1) A partition of nodes to m disjoint communities C_i, $i = 1 \dots m$; and (2) a matrix P of size $m \times m$, which represents the edge probabilities between nodes in different communities, namely, the matrix entry $P_{i,j}$ determines the probability of an edge (v, v') between $v \in C_i$ and $v' \in C_j$. The diagonal elements in P are often much larger than the off-diagonal elements, representing the dense connectivity in a community, compared to the intra-connectivity between communities.

(2) Preferential attachment (PA) networks exhibit a node-degree distribution that follows a power-law (scale-free), like those found in many real-world networks. We use the dual Barbarsi-Albert model (Moshiri, 2018), an extension to the popular Barabasi-Albert model (Barabási, 1999), which allows for continuously varying the mean node degree. The node

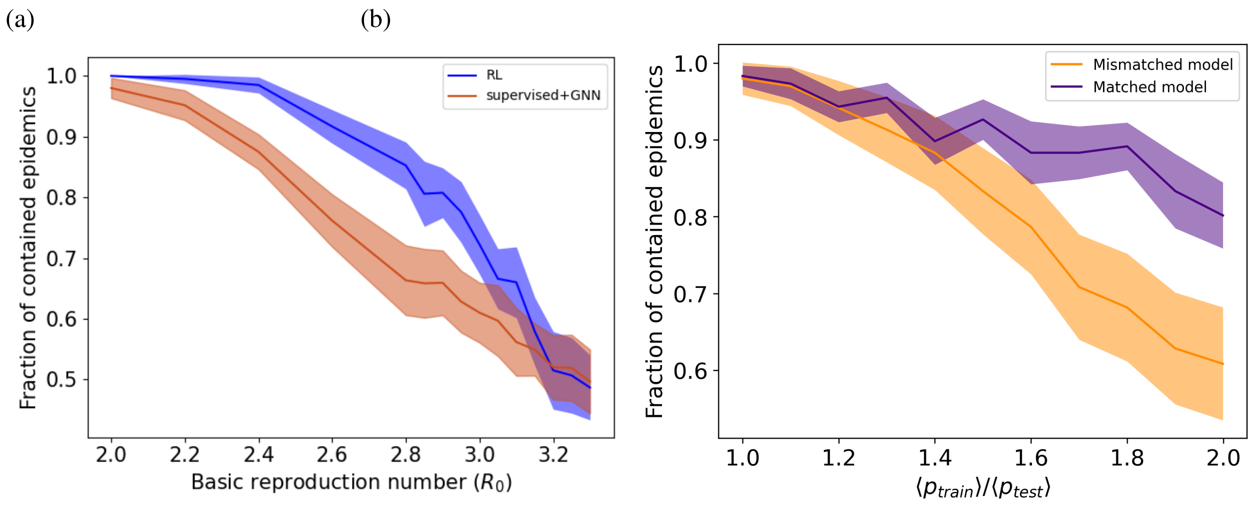


Figure S1. **Stability analysis: (a)** The contained epidemic fraction as a function of the basic reproduction number R_0 on a PA network. RLGN outperforms SL over a large range of R_0 values. **(b)** Stability against test-time shift in transmission probability. *Orange:* The performance of RLGN deteriorates when the mean transmission probability at test time is higher more than 40% than train time. *Purple:* As a baseline, training and testing with the same higher transmission probability.

degree of the resulting network has a power-law distribution.

(3) Contact-tracing networks. We received anonymized high-level statistical information about real contact tracing networks that included the distribution of node degree, transmission probability and mean number of interactions per day, collected during April 2020.

Fig. S2(b) presents the degree distribution in this data, and the transmission probability is presented in Fig. S2(a). The latter was derived based on the contact properties, such as the length and the proximity of the interaction. On average, 1.635 ± 0.211 interactions with a significant transmission probability were recorded per-person per-day. We generated random networks based on these distributions using a configuration model framework (Newman, 2010). The fitted model for the degree distribution is a mixture of a Gaussian and a power-law distribution

$$P(degree = x) = 2.68 \cdot \mathcal{N}(-4.44, 11.18) + 3.2 \cdot 10^{-3} \cdot x^{-0.36}. \tag{4}$$

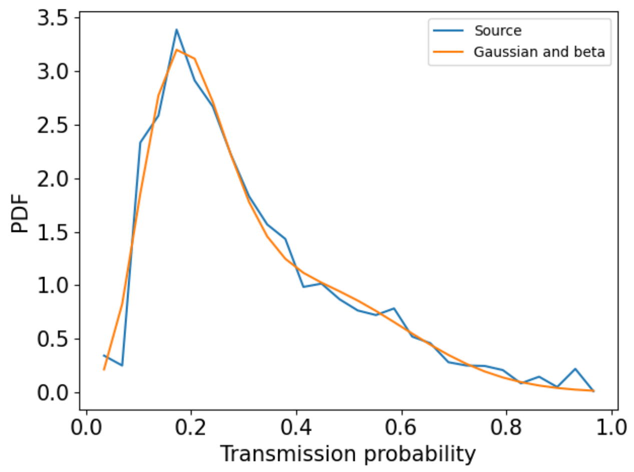
The fitted model for the transmission probability is a mixture of a Gaussian and a Beta distribution

$$P(p_e = x) = 0.47 \cdot \mathcal{N}(0.41, 0.036) + 0.53 \cdot Beta(5.05, 20.02). \tag{5}$$

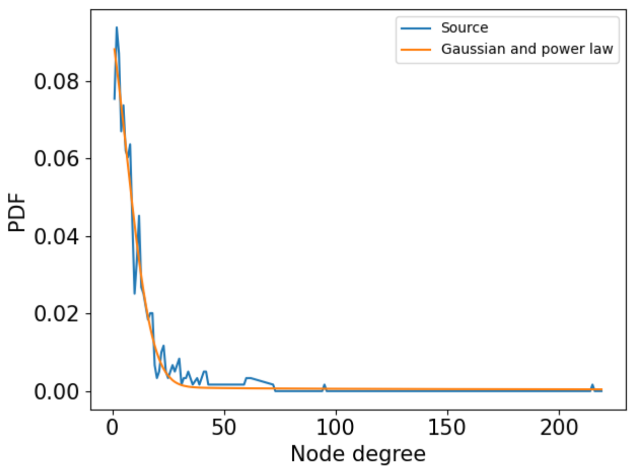
Evaluation on CT data. Due to privacy constraints, we did not have access to "live" CT graphs. We used these statistics to generate topologically similar synthetic graphs. Likewise, we simulated activity patterns based on activity statistics of the real CT network.

D.2. Epidemic test prioritization baselines

A. Preprogrammed Heuristics. The most prevalent baseline, used in practice nowadays in a few countries and circumstances, is based on the proximity of a node to infectious node. We compare with two such methods to ran k nodes. **(1) Infected neighbors.** Rank nodes based on the number of known infected nodes in their 2-hop neighborhood (neighbors and their neighbors). Each node v is assigned a tuple $(I_v^{(1)}, I_v^{(2)})$, and tuples are sorted in a decreasing lexicographical order. A similar algorithm was used in (Meirom et al., 2015; 2018) to detect infected nodes in a noisy environment, and its error was shown to vanish asymptotically in the network size. **(2) Probabilistic risk.** Each node keeps an estimate of the probability it is infected at time $t-1$. To estimate infection probability at time t, beliefs are propagated from neighbors, and dynamic programming is used to analytically solve the probability update. See Appendix E for details. **(3) Degree centrality.** In this baseline high degree nodes are prioritized. This is an intuitive heuristic and it is used frequently (Salathé & Jones, 2010). It was found empirically to provide good results (Sambaturu et al., 2020). **(4) Eigenvalue centrality.** Another common



(a)



(b)

Figure S2. Statistics of a real-world contact-tracing graph. (a) The empirical transition probability $P(p_e)$ on a contact tracing network and our suggested curve fit. (b) The degree distribution on the contact tracing network, along with its fit.

approach it to select nodes using spectral graph properties, such as the eigenvalue centrality (e.g., (Preciado et al., 2014; Yang et al., 2020)).

The main drawback of these heuristic algorithms is that they do not exploit all available information about dynamics. Specifically, they do not use negative test results, which contain information about the true distribution of the epidemic over network nodes.

B. Supervised learning. Algorithms that learn the risk per node using features of the temporal graph, its connectivity and infection state. Then, k nodes with the highest risk are selected. **(5) Supervised (vanilla).** We treat each time step t and each node v_i as a sample, and train a 3-layer deep network using a cross entropy loss against the ground truth state of that node at time t. The input of the DNN has two components: A static component described in Section 4.1, and a dynamic part that contains the number of infected neighbors and their neighbors (like #1 above). Note that the static features include the, amongst other features, the degree and eigenvector centralities. Therefore, if learning is successful, this baseline may derive an improved use of centralities based on local epidemic information. **(6) Supervised (+GNN).** Like #5, but the input to the model is the set of all historic interactions of v_i's and its d-order neighbours and their time stamps as an edge feature. The architecture is a GNN that operates on node and edge features. We used the same ranking module as our GNN framework, but the output probability is regarded as the probability that a node is infected. **(7) Supervised (+weighted degree).** Same as #6, but the loss is modified and nodes are weighted by their degree. Indeed, we wish to favour models that are more accurate on high-degree nodes, because they may infect a greater number of nodes. **(8) Supervised (+weighted degree +GNN).** Like #6 above, using degree-weighted loss like #7.

The supervised algorithm is trained to optimize the model M by minimizing the cross entropy loss function at every time step t between the predicted probability that node v will be infected $\hat{y}_v(t)$ to its groundtruth state $y_v(t)$:

$$-\frac{1}{N}\sum_v w_v\, CrossEntropy(y_v(t), \hat{y}_v(t)) \tag{6}$$

In the unweighted cross entropy loss, all nodes are weighted equally, $w_v = 1$. We can strengthen this baseline by noting that we would like the model to be more accurate on high degree nodes, as they may infect a greater number of nodes. Hence, we weigh the contribution of a node to the loss term by its degree, $w_v = \deg(v)$. We refer to the latter algorithm as a degree weighted SL.

The main drawback of the supervised algorithms is that they optimize a myopic loss function. Therefore, they are unable to optimize the long term objective and consider the strategic importance of nodes.

Metrics. The end goal of quarantining and epidemiological testing is to minimize the spread of the epidemic. As it is unreasonable to eradicate the epidemic using social distancing alone, the hope is to "flatten the curve", namely, to slow down the epidemic progress. Equivalently, for a simulation with fixed length, the goal is to reduce the number of infected nodes. In addition to the **%healthy** metric, defined in Sec. 5, we consider an additional metric, **%contained**: The probability of containing the epidemic. This was computed as the fraction of simulations having cumulative infected nodes smaller than a fraction α. We focus on this metric because it captures the important notion of the capacity of a health system. In the 2-community setup, where each community has half of the nodes, a natural choice of α is slightly greater than 0.5, capturing those cases where the algorithm contains the epidemic within the infected community. In all the experiments we set $\alpha = 0.6$. The only exception is the three-communities experiments, in which we set the bar slightly higher than $1/3$, and fixed $\alpha = 0.4$.

D.3. Epidemic Test Prioriritzation - Additional Experiments

Epidemic slowdown. We investigated the progression of the epidemic under either RLGN or supervised+GNN algorithms. Figure S4 shows that the epidemic spread speed is slower under the RLGN policy in all graphs. In general, there are two extreme configuration regimes. First, the "too-hard" case, when the number of tests is insufficient to block the epidemic, and second, the "too-easy" case when there is a surplus of tests such that every reasonable algorithm can contain it. The more interesting case is the intermediate regime, where some algorithms succeed to delay the epidemic progression, or block it completely, better than other algorithms. Fig. S4(a) illustrates the case where the number of tests is insufficient for containing the epidemic, for all algorithms we tested. In Fig. S4(b), the number of tests is insufficient for SL to block the epidemic. However, with same number of tests, RLGN algorithm successfully contains the epidemic. Fig. S4(c) presents an interesting case where RLGN slows down the epidemic progression and reduces the number of total number of infected node, compared with SL, but RL does not contain it completely.

Epidemiological model variations. Figure S1(b) depicts a robustness analysis of RLGN for variations in the epidemiological model. One of the most difficult quantities to assess is the probability for infection per social interaction. Figure S1(b) shows that the trained model can sustain up to $\sim 40\%$ deviation at test time in this key parameter.

Graph size variations. We have tested the robustness of our results to the underlying graph size. Specifically, we compare the two best algorithms RLGN (#8) and SL+GNN (#4), using graphs with various sizes, from 300 nodes to 1000 nodes. Table S6 compares RLGN with the SL+GNN algorithm on preferential attachment (PA) networks (mean degree $= 2.8$). We provide results for various sizes of initial infection i_0 and number of available tests k at each step. The experiments show that there is a considerable gap between the performance of the RL and the second-best baseline. Furthermore, RLGN achieves better performance than the SL+GNN algorithm with 40%-100% more tests. Namely, it increases the effective number of tests by a factor of $\times 1.4 - \times 2$.

Initial infection size. We also tested the sensitivity of the results to the relative size of the initial infection. Table S6 shows results when 4% of the the network was initially infected, as well as for 7.5% and 10%. The results show that RLGN outperforms the baselines in this wide range of infection sizes.

D.4. Influence Maximization - Additional Experiments

We presented two natural extensions to the Independent Cascades model. In Table 3 in the main paper we presented the results of the IC(geometric) model. Table S1 presents the results of the IC(constant) model. In both cases, RLGN often outperform the state-of-the-art algorithms. In both simulations, the neighbor influence probability was $p = 0.25$, the agent's success probability for setting a node to *Influenced* state was $q = 0.3$, and the probability that an influenced node will reveal its state was $\eta = 0.25$.

We follow with a comparison of the performance of the various algorithms on the Linear Threshold model (Table S2). Here, RLGN clearly outperformed the other baselines. The main reason to the increased gap is that a Linear Threshold model contains more parameters, as each edge and is associated with a random weight and each node is assigned a peer resistance value. RLGN, as a trainable model, is able to uncover relevant patterns, while the other algorithms fail to do so. Table S2 also shows that the RLGN performs better thatn the baselines over a variety of the dynamic model parameters.

D.5. Ablation Studies

Mapping scores to action distribution. We compare the performance of our score-to-probability function (calibrated-scores) to the popular softmax (Boltzmann) distribution. In practice, in most instances, we were unable to train a model using

	gemsec-RO	Email	ca-HepTh	ca-GrQc
LIR	37.5 ± 1	73.2 ± 0.3	58.0 ± 0.3	47.4 ± 0.4
LIR (filtered)	36.3 ± 1	73.6 ± 0.3	58.3 ± 0.4	48.7 ± 0.4
Degree	25.4 ± 1	74.0 ± 0.3	58.0 ± 0.3	48.5 ± 0.4
Degree Discounted	25.9 ± 1	70.4 ± 0.5	58.2 ± 0.5	48.4 ± 0.4
Eigenvector	24.8 ± 1	$\mathbf{74.3 \pm 0.3}$	48.6 ± 0.6	48.5 ± 0.4
RLGN	$\mathbf{71.6 \pm 1}$	$\mathbf{74.2 \pm 0.2}$	$\mathbf{59.5 \pm 0.5}$	48.9 ± 0.3

Table S1. The percentile of influenced nodes on the real-world networks in the Influence Maximization setup. IC(constant) was used as the dynamical model. Each episode lasted 15 steps, and each result represents the mean *%influenced* value after 300 episodes. Results on the Portland network were omitted as all algorithms were able to achieve $> 98\%$ influenced share on this network.

	ca-GrQc	gemsec-RO	ca-HepTh	Email
LIR	3.8 ± 0.1	0.014 ± 0.006	1.4 ± 0.1	14.2 ± 0.4
LIR(filtered)	5.2 ± 0.1	0.019 ± 0.001	3.1 ± 0.2	15.8 ± 0.3
Degree	4.5 ± 0.1	0.007 ± 0.002	1.4 ± 0.1	17.3 ± 0.3
Degree Discounted	4.7 ± 0.1	0.007 ± 0.004	1.14 ± 0.1	14.0 ± 0.4
Eigenvector	4.7 ± 0.1	0.008 ± 0.004	0.42 ± 0.01	18.4 ± 0.3
RLGN	$\mathbf{7.1 \pm 0.2}$	$\mathbf{1.98 \pm 0.06}$	$\mathbf{5.7 \pm 0.2}$	$\mathbf{21.8 \pm 0.1}$

	ca-GrQc	gemsec-RO	ca-HepTh	Email
LIR	5.5±0.1	0.028±0.001	2.8±0.2	18.9±0.5
LIR(filtered)	6.9±0.1	0.034±0.002	5.6±0.2	21.2±0.4
Degree	6.2±0.1	0.01±0.001	2.8±0.2	23.6±0.3
Degree Discounted	6.1±0.1	0.009±0.001	3.2±0.2	18.9±0.4
Eigenvector	6.2±0.1	0.011±0.001	0.58±0.03	24.2±0.3
RL+GNN	$\mathbf{10.7 \pm 0.1}$	$\mathbf{4.3 \pm 0.1}$	$\mathbf{12.2 \pm 0.2}$	$\mathbf{26.5 \pm 0.1}$

Table S2. The percentile of influenced nodes on the real-world networks in the Influence Maximization setup. Linear threshold was used as the dynamical model. In the top table, the peer resistance value z_v was sampled uniformly from $[0.5, 1.5]$ and in the lower table it was sampled uniformly from $[0.4, 1.4]$. Each edge was assigned a uniform random weight $[0, 1]$. Each episode lasted 20 steps, and each result represents the mean *%influenced* value after 300 episodes.

	$\%contained$	# training epochs
Sigmoid	0.84 ± 0.05	1210
GRU	0.91 ± 0.03	810
L_2 norm.	$\mathbf{0.93 \pm 0.02}$	**500**

Table S3. Training time and fraction of contained epidemic for three normalization schemes. The L_2 normalization scheme is fastest and achieves the best performance.

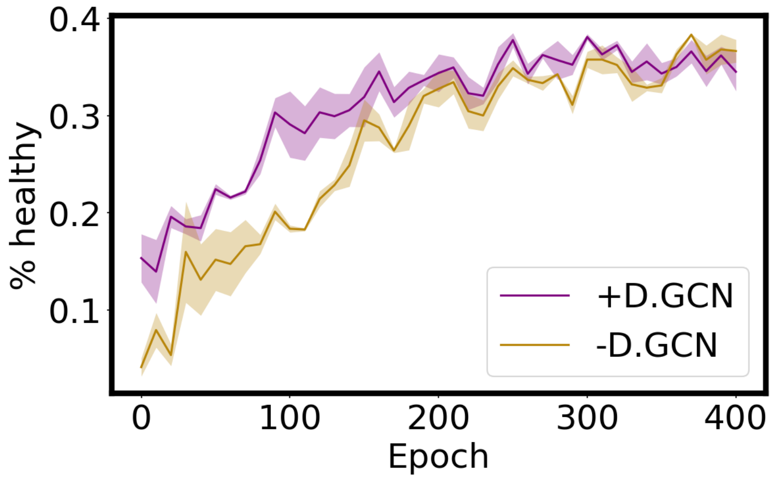


Figure S3. Training curves with and without a local-diffusion GCN (D. GCN) module on a preferential attachment network.

the softmax distribution as the neural network weights diverge. Fig. S5 presents the training curve in one of the few instances that did converge. It is clear that the model was not able to learn a useful policy while using the calibrated-scores probability function resulted in a corresponding value of more than 0.75. We also compare our approach with the recent escort transform (Mei et al., 2020), a state-of-the-art score-to-probability method. As shown in Fig. S5, our method outperforms the escort transform in this problem.

Normalization in scale-free networks. We compared the suggested normalization to a number of other alternative normalization methods. (1) Applying a sigmoid layer after the hidden state update module G. (2) Replace the hidden state update module with a GRU layer. (3) Apply L_2 normalization to each feature vector $h_v(t)$ (similarly to (Hamilton et al., 2017)) (4) Normalize the feature vector matrix by its L_2 norm. These four normalization schemes span three different types of normalization: single-feature normalization (1+2), vector normalization (3), and matrix normalization (4).

Table S3 presents the score after training and the number of training steps required to complete training. Method (4) was unstable and training did not converge, therefore it was omitted from the table. The main reason for the training time difference is that without normalization, the DNN weights' magnitude increases. In a GRU module, or with a direct application of a sigmoid layer, the features pass through a sigmoid activation function. When the magnitude of the input to this layer is large, the gradient is very small due to the sigmoid plateau. This substantially slows down the learning process.

	#Nodes	#Edges
CA-GrQc	5242	14496
Montreal	103425	630893
Portland	10000	199167
Email	32430	54397
GEMSEC-RO	41773	222887

Table S4. Number of edges and nodes in the large-scale datasets

%CONTAINED	2 COMMUNITIES	3 COMMUNITIES
TREE-BASED MODEL	15 ± 35	0 ± 0
COUNTER MODEL	19 ± 39	1 ± 4
DEGREE CENTRALITY	23 ± 1	0 ± 0
EIGENVECTOR CENTRALITY	19 ± 3	0 ± 0
SUPERVISED (VANILLA)	24 ± 11	2 ± 2
SUPERVISED +GNN	27 ± 10	2 ± 2
SUPERVISED +DEGREE	29 ± 10	1 ± 2
SUPERVISED +GNN+DEG	24 ± 10	2 ± 02
RLGN (VANILLA)	66 ± 10	7 ± 5
RLGN FULL (OURS)	**88±1**	**53±13**

Table S5. Probability (in %) of containing an epidemic in community-based networks. Each community has 30 densely connected nodes, and the test budget is $k = 2$.

$n = 300$	Init. infection size 5%		Init. infection size 7.5%		Init. infection size 10%	
	%healthy	%contained	%healthy	%contained	%healthy	%contained
SL, $k = 1\%$	27 ± 2	15 ± 5	21 ± 2	4 ± 2	18 ± 1	1 ± 1
SL, $k = 1.33\%$	41 ± 3	37 ± 6	27 ± 2	12 ± 4	24 ± 2	6 ± 3
SL, $k = 2\%$	66 ± 4	76 ± 6	48 ± 3	55 ± 7	37 ± 2	32 ± 6
RLGN, $k = 1\%$	50 ± 2	78 ± 7	43 ± 2	58 ± 1	40 ± 1	48 ± 6

$n = 500$	Init. infection size 5%		Init. infection size 7.5%		Init. infection size 10%	
	%healthy	%contained	%healthy	%contained	%healthy	%contained
SL, $k = 1\%$	24 ± 2	7 ± 4	20 ± 1	2 ± 1	19 ± 1	0 ± 1
SL, $k = 1.6\%$	48 ± 3	54 ± 6	35 ± 2	27 ± 7	29 ± 1	11 ± 1
SL, $k = 2\%$	67 ± 3	83 ± 5	46 ± 2	53 ± 4	38 ± 2	37 ± 7
RLGN, $k = 1\%$	52 ± 1	97 ± 2	44 ± 2	75 ± 11	42 ± 1	66 ± 6

$n = 1000$	Init. infection size 5%		Init. Infection size 7.5%		Init. infection size 10%	
	%healthy	%contained	%healthy	%contained	%healthy	%contained
SL, $k = 1\%$	25 ± 2	5 ± 3	21 ± 1	0 ± 1	19 ± 1	0 ± 0
SL, $k = 1.5\%$	42 ± 2	49 ± 6	30 ± 1	10 ± 3	27 ± 1	4 ± 2
SL, $k = 2\%$	66 ± 1	84 ± 5	45 ± 2	59 ± 5	37 ± 1	30 ± 1
RLGN, $k = 1\%$	52 ± 1	97 ± 2	44 ± 2	75 ± 11	42 ± 1	66 ± 6

Table S6. A comparison between RLGN and SL+GNN (the best epidemic test prioritization baseline). RLGN performance is highlighted. The number of additional resources needed to surpass the RLGN performance in a given metric is also highlighted. In many cases, even using SL+GNN with twice as many resources than RLGN performs worse than RLGN. The evaluation was performed on a preferential attachment network with mean degree 2.8. The number of nodes is indicated at the top of each table.

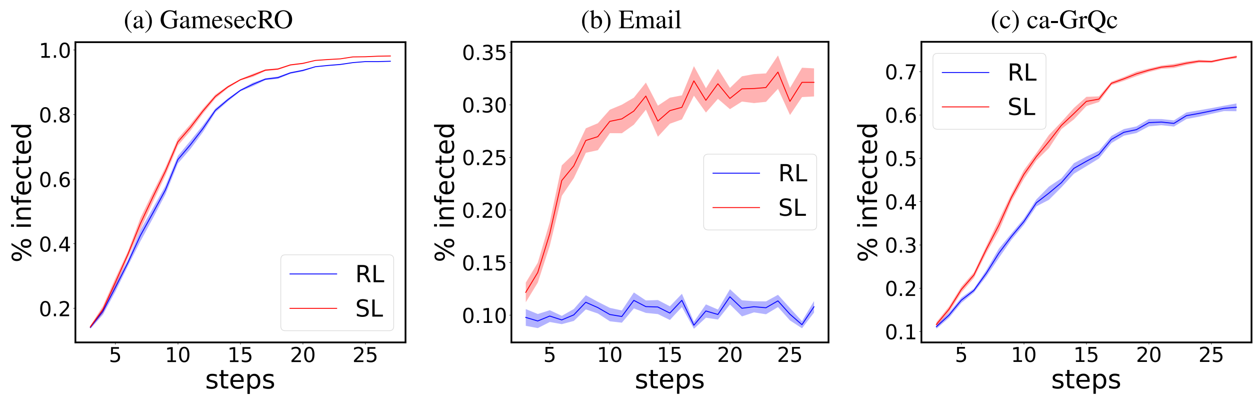


Figure S4. The fraction of infected nodes as a function of time step t. Shaded areas represent one standard deviation around the mean. (a) The epidemic propagation on an online social (gemsec-RO) (b) The epidemic propagation on an email network. (c) The epidemic propagation on the collaboration graph ca-GrQc. In all cases the epidemic propagates more slowly under RLGN compared with the best baseline (supervised+GNN, #4).

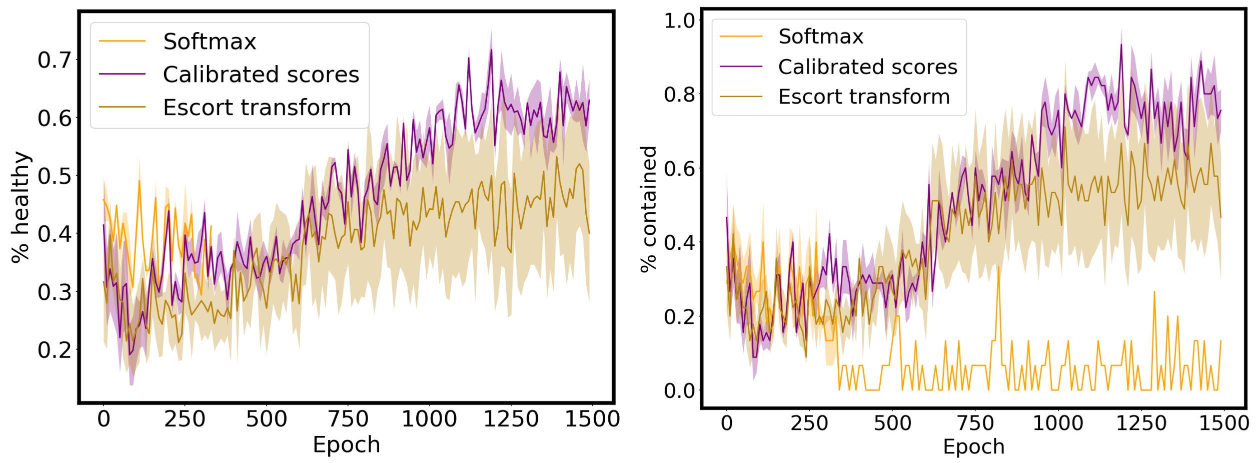


Figure S5. Fraction of contained epidemics and %healthy during training in a preferential attachment model with 200 nodes and a mean degree 2.8. For non-normalized mapping, only one of the three seeds in the softmax distribution simulation completed training due to numerical instability. No stability issues were observed when using the calibrated scores normalization scheme described by Eq. (3).

D.6. Network architecture

The architecture of the ranking module is shared by algorithms #4, #6 and #8 with slight variations indicated below.

Input. We encode the dynamic node features $\zeta_v^d(t)$ as follows. **Epidemic test priorization**: A one hot hot vector of dimension 4. Each of the first three elements corresponds to one of the three mutually exclusive options, which depends on the previous step: untested, tested positive, tested negative. The last entry indicates whether a node was found positive in the past. **Influence maximization**: A one hot hot vector of dimension 3. Similarly, the first two elements indicate whether a node has indicates it is influenced in the previous step or not, and the last element whether it indicated so in the past. The static node features, $\zeta_v^s(t)$, are common to both problems. As described in the main paper, topological graph centralities (betweenness, closeness, eigenvector, and degree centralities) and random node features. The graph centralities are standard metrics, and were calculated using NetworKit. An ablation study showed a maximal performance difference of 2% between different centralities subsets and was statistically insignificant.

Local Diffusion GNN. This module M_e is composed of a single graph convolutional layer. The input features are the last time step node features. The number of output features is 64.

Long Range Information GNN. Each message passing module M^l contains one hidden layer, where the number of hidden features is 64. After both the hidden and the last layer we apply a leaky ReLu layer with leakage constant 0.01. After aggregating the result using the addition aggregation function, we apply an additional MLP with one layer (linear+ReLu) on the resulting feature vector. The number of output features is 64. The value of τ, the information window size, was 7 in all our experiments.

We experimented with the numbers of stacked modules l (layers). We found that $l = 3$ performed slightly better than $l = 2$ but training was considerably slower because the batch size had to be reduced. We therefore used $l = 2$ in all experiments reported.

Hidden state update. The hidden state MLP G is composed of a single linear layer follows by a ReLu activation layer. To keep the resulting hidden feature vector (of dimension 64) norm under check, an additional normalization scheme is then applied. This module was replaced by a GRU layer in the ablation studies.

Output layer. The last module is a single linear layer, with an output dimension as the number of the nodes in the graph.

Learning framework. We used Pytorch (Paszke et al., 2017) and Pytorch Geometric (Fey & Lenssen, 2019) to construct the ranking module. We used ADAM with default parameters as our optimizer.

D.7. Training protocol

Initializaition. The initialization depends on the problem setup. Each influenced node signal with probability q at each turn. In the epidemic tests prioritization setup we trained the RL and SL algorithms by generating random networks and initializing each network by selecting for each instance a random subset of m_0 infected nodes. We propagate the epidemic until it spans at least i_0 infected nodes (for at least t_0 steps), and randomly detect a subset of the infected nodes of size $k_0 < i_0$. The initialization for the Influence Maximization setup is simpler, and conclude in generating a random network.

At each step, in all algorithms but RL, we pick the top k rated nodes. In RL, we perform the same procedure during the evaluation phase, while during training we sample k nodes using the score-to-probability distribution.

Each model was trained for at most 1500 episodes, but usually, training was completed after 1000 episodes. Each episode contained 1024 steps, collected by 4 different workers. As our network contains a recurrent module, we propagate each sample in the buffer for three steps, in a similar fashion to R2D2.

For each setup we described, at least three models were trained using different seeds, and the results are the average over the performance of all models. The errors are the standard deviation of the mean. over at least 100 evaluation episodes for each model.

Each episode lasted for 25 steps, each corresponds conceptually to a day. The transition time from the latent to the infectious state was normally distributed with a mean of two steps and a standard deviation of 1 step, corresponding to real-world

minimal #propagation steps (t_0)	4
minimal #infected component size (i_0)	communities: 4 (same community)
	preferential attachment: 5%
	contact tracing: 7%
Learning rate	$3 \cdot 10^{-4}$
λ	0.97
γ	0.99
Entropy loss weight	0.01
Value loss weight	0.5
Probability distribution of $e \in \mathcal{E}(t)$	$U[0.2, 0.6]$
Batch size	256 (128 if #nodes> 200)
3-communites SBM matrix	$\begin{pmatrix} 0.6 & 0.001 & 0 \\ 0.001 & 0.6 & 0.001 \\ 0 & 0.001 & 0.6 \end{pmatrix}$
2-communites SBM matrix	$\begin{pmatrix} 0.6 & 0.0022 \\ 0.0022 & 0.6 \end{pmatrix}$

Table S7. Parameters table

values. The advantage was calculated using the Generalized Advantage framework with parameters $\gamma = 0.99, \lambda = 0.97$.

Table S7 presents the simulation parameters used in the main paper. We shall make the repository and code available online.

E. The tree model baseline

In this appendix We describe our tree model baseline (algorithm #1). Consider an epidemic propagating on a tree, and assume there is a single initially infected node (”patient zero”). In this case, there is a single path from the infection source to every node in the graph and we can we can analytically solve for the probability a node is infected, given that the root of the tree was infected at time t_0. This model is useful when the underlying network is locally a tree, i.e, that for every new infected node v there is w.h.p just one node which may have infected it.

We start with a simple case.

E.1. Simple case: No latent state

Let us first consider a simple model in which the epidemic spreads on a tree like structure with a single epidemic source, a.k.a. patient-zero, as the root. For now, let us assume there is no latent state.

Our goal is to calculate the probability that a node n will be infected at time T

$$F_n(T) \triangleq \Pr\left(ST_n(T) = \mathcal{I} | ST_r(0) = \mathcal{I}\right)$$

For every node j there is a single path from the node to the root, denoted by r. Let us assume the path is $\{y_0 = r, y_1, y_2, ..y_{n-1}, y_n = j\}$. Assume that in $[0, T]$ a sequence of interactions between node y_n and y_{n-1} occurred at discrete times $(t_1, t_2, ...t_m)$, and that each interaction is characterized by an infection probability $(p_1, p_2, ...p_m)$. We evaluate $F_n(T)$ by induction. For abbreviation, we write $ST_{y_i}(t) = Y_i(t)$ and denote the event $ST_r(0) = \mathcal{I}$ as A.

Our key result is that The state of node n at the time of interaction m is a function of its state at penultimate interaction time $F_n(t_{m-1})$, the interaction transmission probability p_m, and the predecessor node $n-1$ state at time m, $F_n(t_{m-1})$.

$$\begin{aligned} F_n(t_m) &= F_n(t_{m-1}) + p_m\left(F_{n-1}(t_m) - F_n(t_{m-1})\right) \\ &= p_m F_{n-1}(t_m) + F_n(t_{m-1})\left(1 - p_m\right) \end{aligned}$$

The first term is the probability to get infected at the m interaction, and the second term is the probability to get infected before hand. We shall now prove this result.

Proof. We can write the conditional probability using a graphical model decomposition and obtain

$$\begin{aligned}
&\Pr\left(Y_n(T)=\mathcal{I}|A\right)= \\
&\Pr\left(Y_n(t_m)=\mathcal{I}|Y_{n-1}(t_m)=\mathcal{I},A\right)\Pr\left(Y_{n-1}(t_m)=\mathcal{I}|A\right)= \\
&\Pr\left(Y_n(t_m)=\mathcal{I}|Y_{n-1}(t_m)=\mathcal{I},A\right)F_{n-1}(t_m)
\end{aligned} \tag{7}$$

since if the ancestor node is not in an infectious state, the decedent can not be infected. Denote the indicator that interaction l was able to transmit the epidemic as I_l. We have,

$$\begin{aligned}
&\Pr\left(Y_n(t_m)=\mathcal{I}|Y_{n-1}(t_m)=\mathcal{I},A\right) && = \\
&\sum_{l=1}^{m}\Pr\left(y_n\text{'s infection time is } t_l|Y_{n-1}(t_m)=\mathcal{I},A\right) && = \\
&\sum_{l=1}^{m}\Pr\left(Y_n(t_{l-1})=\mathcal{H},I_l,Y_{n-1}(t_l)=\mathcal{I}|Y_{n-1}(t_m)=\mathcal{I},A\right)
\end{aligned}$$

As, for an infection event to take place at it must be that node y_{n-1} was infected at t_l, node y_n was healthy beforehand, and that the interaction resulted in an infection. We can now write this as

$$\begin{aligned}
&\Pr\left(Y_n(t_{l-1})=\mathcal{H},I_l,Y_{n-1}(t_l)=\mathcal{I}|Y_{n-1}(t_m)=\mathcal{I},A\right) && = \\
&p_l\Pr\left(Y_n(t_{l-1})=\mathcal{H},Y_{n-1}(t_l)=\mathcal{I}|Y_{n-1}(t_m)=\mathcal{I},A\right) && = \\
&p_l\Pr\left(Y_n(t_{l-1})=\mathcal{H},Y_{n-1}(t_m)=\mathcal{I}|Y_{n-1}(t_l)=\mathcal{I},A\right)\frac{\Pr\left(Y_{n-1}(t_l)=\mathcal{I}|A\right)}{\Pr\left(Y_{n-1}(t_m)=\mathcal{I}|A\right)} && = \\
&p_l\Pr\left(Y_n(t_{l-1})=\mathcal{H}|Y_{n-1}(t_l)=\mathcal{I},A\right)\frac{F_{n-1}(t_l)}{F_{n-1}(t_m)} && = \\
&p_l\left(1-\Pr\left(Y_n(t_{l-1})=\mathcal{I}|Y_{n-1}(t_l)=\mathcal{I},A\right)\right)\frac{F_{n-1}(t_l)}{F_{n-1}(t_m)}
\end{aligned} \tag{8}$$

The transition from the first line to the second is due to the independence of the interaction infection probability with the history of the participating parties. The third line is Bayes' theorem. If a node is infected at time t_l, it will be infected later on at t_m, as expressed in line 4. The last line is the complete probability formula.

We rewrite $\Pr\left(Y_n(t_{l-1})=\mathcal{I}|Y_{n-1}(t_l)=\mathcal{I},A\right)$ as

$$\begin{aligned}
&\Pr\left(Y_n(t_{l-1})=\mathcal{I}|Y_{n-1}(t_l)=\mathcal{I},A\right)= \\
&\frac{\Pr\left(Y_n(t_{l-1})=\mathcal{I}|A\right)-\Pr\left(Y_n(t_{l-1})=\mathcal{I},Y_{n-1}(t_l)=\mathcal{H}|A\right)}{\Pr\left(Y_{n-1}(t_l)=\mathcal{I}|A\right)}= \\
&\frac{\Pr\left(Y_n(t_{l-1})=\mathcal{I}|A\right)}{\Pr\left(Y_{n-1}(t_l)=\mathcal{I}|A\right)}= \\
&\frac{F_n(t_{l-1})}{F_{n-1}(t_l)}
\end{aligned}$$

The transition from the first line to the second line is a complete probability transition. The third line is due to the fact that if y_{n-1} was not infected at time t_l, clearly y_n could not be infected before t_l. We have

$$\begin{aligned}
F_n(t_m)=\Pr\left(Y_{n-1}(t_m)=\mathcal{I}|A\right)&=\sum_{l=1}^{m}p_l\left(1-\frac{F_n(t_{l-1})}{F_{n-1}(t_l)}\right)\frac{F_{n-1}(t_l)}{F_{n-1}(t_m)}F_{n-1}(t_m) \\
&=\sum_{l=1}^{m}p_l\left(F_{n-1}(t_l)-F_n(t_{l-1})\right)
\end{aligned}$$

Therefore, given $F_{n-1}(t_l)$ for all $l \in \{1..n-1\}$ and $F_n(t_l)$ for all $l \in \{1..n\}$, we can directly calculate the infection probabilities, given the initial condition: $F_i(0) = \delta_{i,0}$.

We can write the partial density function of $F_i(t_l)$ as $f_i(t_l) = F_i(t_l) - F_i(t_{l-1})$, and obtain: $f_n(t_m) = p_m\left(F_{n-1}(t_m) - F_n(t_{m-1})\right)$. This allows us to write this with an intuitive formulation

$$\begin{aligned} F_n(t_m) &= F_n(t_{m-1}) + p_m\left(F_{n-1}(t_m) - F_n(t_{m-1})\right) \\ &= p_m F_{n-1}(t_m) + F_n(t_{m-1})\left(1 - p_m\right) \end{aligned}$$

The first term is the probability to get infected at the m interaction, and the second term is the probability to get infected before hand.

□

E.2. Full analysis with latent states

We now discuss the case where a node can be in a latent state. The main difference is that the complement of the infectious state is composed of two states, healthy $\mathcal{H}$, and latent $\mathcal{L}$. We shall denote all the non-infecting states as $\mathcal{H}^+ = \{\mathcal{H}, \mathcal{L}\}$ and all the infected states as $\mathcal{I}^+ = \{\mathcal{I}, \mathcal{L}\}$, and sometime abuse the notation by writing $S_i(t) = \mathcal{H}^+$. We denote the transmission delay from the latent to infectious state as $L(\tau)$.

As before, we are interested in the probability that

$$\Pr\left(Y_n(T) = \mathcal{I}^+ | S_r(0) = \mathcal{I}\right)$$

The derivation below shows that, similar to the previous case, we can solve for this probability using dynamic programming. The end result is that

$$\Pr\left(Y_n(T) = \mathcal{I}^+ | ST_r(0) = \mathcal{I}\right) = \sum_{l=1}^{m} p_l\left(F_{n-1}(t_l) - F_n(t_{l-1}) - \Pr\left(Y_n(t_{l-1}) = \mathcal{L}|A\right)\right) ,$$

with

$$\Pr\left(Y_n(t_l) = \mathcal{L}|A\right) = \sum_{t_i < t_l} \left(1 - L(t_i - t_l)\right) q_n(t_i)$$

and

$$q_n(t_m) = p_m\left(F_{n-1}(t_m) - F_n(t_{m-1}) - \Pr\left(Y_n(t_{l-1}) = \mathcal{L}|A\right)\right).$$

Therefore, as before, given $F_{n-1}(t_m)$ and $q_n(t_i)$ for all $i < m$, we can propagate and calculate $q_n(t_m)$ and $F_n(t_m)$.

Proof. We start with an equation equivalent to Eq. 7,

$$\begin{aligned} &\Pr\left(Y_n(T) = \mathcal{I}^+ | A\right) = \\ &\Pr\left(Y_n(t_m) = \mathcal{I}^+ | Y_{n-1}(t_m) = \mathcal{I}, A\right) \Pr\left(Y_{n-1}(t_m) = \mathcal{I}|A\right) = \\ &\Pr\left(Y_n(t_m) = \mathcal{I}^+ | Y_{n-1}(t_m) = \mathcal{I}, A\right) F_{n-1}(t_m) \end{aligned}$$

where we kept the definition of $F_j(t)$. Therefore, almost identically,

$$\begin{aligned} &\Pr\left(Y_n(t_m) = \mathcal{I}^+ | Y_{n-1}(t_m) = \mathcal{I}, A\right) = \\ &\sum_{l=1}^{m} \Pr\left(y_n\text{'s infection time is } t_l | Y_{n-1}(t_m) = \mathcal{I}, A\right) = \\ &\sum_{l=1}^{m} \Pr\left(Y_n(t_{l-1}) = \mathcal{H}, I_l, Y_{n-1}(t_l) = \mathcal{I} | Y_{n-1}(t_m) = \mathcal{I}, A\right). \end{aligned}$$

Eq. 8 follows up to the last line, where:

$$\begin{aligned}
&\Pr\left(Y_n(t_{l-1}) = \mathcal{H}, I_l, Y_{n-1}(t_l) = \mathcal{I} | Y_{n-1}(t_m) = \mathcal{I}, A\right) = \\
&p_l \Pr\left(Y_n(t_{l-1}) = \mathcal{H} | Y_{n-1}(t_l) = \mathcal{I}, A\right) \frac{F_{n-1}(t_l)}{f_{n-1}(t_m)} = \\
&p_l \left(1 - \Pr\left(Y_n(t_{l-1}) = \mathcal{I}^+ | Y_{n-1}(t_l) = \mathcal{I}, A\right)\right) \frac{F_{n-1}(t_l)}{F_{n-1}(t_m)}
\end{aligned}$$

and,

$$\begin{aligned}
&\Pr\left(Y_n(t_{l-1}) = \mathcal{I}^+ | Y_{n-1}(t_l) = \mathcal{I}, A\right) = \\
&\frac{\Pr\left(Y_n(t_{l-1}) = \mathcal{I}^+ | A\right) - \Pr\left(Y_n(t_{l-1}) = \mathcal{I}, Y_{n-1}(t_l) = \mathcal{H}^+ | A\right)}{\Pr\left(Y_{n-1}(t_l) = \mathcal{I} | A\right)} = \\
&\frac{\Pr\left(Y_n(t_{l-1}) = \mathcal{I}^+ | A\right)}{\Pr\left(Y_{n-1}(t_l) = \mathcal{I} | A\right)} = \\
&\frac{\Pr\left(Y_n(t_{l-1}) = \mathcal{I}^+ | A\right)}{F_{n-1}(t_l)}.
\end{aligned}$$

To summarize, we obtain:

$$\begin{aligned}
\Pr\left(Y_n(T) = \mathcal{I}^+ | S_r(0) = \mathcal{I}\right) &= \sum_{l=1}^{m} p_l \left(1 - \frac{\Pr\left(Y_n(t_{l-1}) = \mathcal{I}^+ | A\right)}{F_{n-1}(t_l)}\right) \frac{F_{n-1}(t_l)}{F_{n-1}(t_m)} F_{n-1}(t_m) \\
&= \sum_{l=1}^{m} p_l \left(F_{n-1}(t_l) - \Pr\left(Y_n(t_{l-1}) = \mathcal{I}^+ | A\right)\right) \\
&= \sum_{l=1}^{m} p_l \left(F_{n-1}(t_l) - F_n(t_{l-1}) - \Pr\left(Y_n(t_{l-1}) = \mathcal{L} | A\right)\right)
\end{aligned}$$

Let us denote the probability density function that an infection occurred during interaction m as

$$q_n(t_m) = \Pr\left(Y_n(t_m) = \mathcal{I}^+ | A\right) - \Pr\left(Y_n(t_{m-1}) = \mathcal{I}^+ | A\right). \tag{9}$$

We have,

$$q_n(t_m) = p_m \left(F_{n-1}(t_m) - F_n(t_{m-1}) - \Pr\left(Y_n(t_{l-1}) = \mathcal{L} | A\right)\right).$$

The transition from the latent state to the infected state follows:

$$F_n(t_l) = \Pr\left(Y_n(t_l) = \mathcal{I} | A\right) = \sum_{t_i < t_l} L(t_i - t_l) q_n(t_i) \tag{10}$$

while

$$\Pr\left(Y_n(t_l) = \mathcal{L} | A\right) = \sum_{t_i < t_l} \left(1 - L(t_i - t_l)\right) q_n(t_i). \tag{11}$$

Therefore, given $F_{n-1}(t_m)$ and $q_n(t_i)$ for all $i < m$, we can propagate and calculate $q_n(t_m)$ and $F_n(t_m)$. $\square$