NetRL: Task-aware Network Denoising via Deep Reinforcement Learning

Jiarong Xu, Yang Yang, Shiliang Pu, Yao Fu, Jun Feng, Weihao Jiang, Jiangang Lu and Chunping Wang

Abstract—Network data is mostly hard to obtain and error-prone. However, most existing works assume that the studied network represents a perfect and complete picture of topological structure; nevertheless, it is rarely the case in real-world situations. Such studies, performing downstream applications (e.g., vertex classification, link prediction, etc.) directly on original networks, will suffer greatly due to the noise and deteriorate the application performance. In this paper, we propose NetRL, a novel method for network denoising, that works by creating missing edges and removing incorrect edges from a noisy network, thereby improving its quality and representative power. In particular, NetRL turns the problem of network denoising into edge sequences generation, which can be formulated as a Markov Decision Process. By doing this, NetRL takes the complex long-term dependency between edge creations into consideration, i.e., the existence of an edge depends on which edges have been generated so far. It further takes advantage of downstream task to guide the network denoising process, by providing a deep reinforcement learning framework to conduct direct optimization on this task-specific objective. As a result, NetRL ensures that the denoised network not only satisfies the topological property of the original network, but also improves the performance of the downstream application. Experimental results on real-world networks show that, comparing with several baseline methods, NetRL can denoise networks effectively with better performance for vertex classification. Meanwhile, NetRL can better preserve original network's properties (e.g., degree distribution and clustering coefficient. Our implementation is available at: https://github.com/galina0217/NetRL.

Index Terms—Network Denoising, Reinforcement Learning.

1 Introduction

A Network is a widely used data structure for describing and modeling complex systems, such as social networks, biochemical networks, information networks, etc.. There has been an explosion of empirical studies over the years focusing on network data.

One fundamental issue is that network data is hard to obtain and error-prone in reality. Most previous studies have assumed that the obtained network represents a perfect and complete picture of the vertices and edges.

However, most real data only depicts an imperfect picture of network structure, which may be blamed on experimental error, incomplete sampling, human subjectivity of some kind. For example, to conduct a mobile network from call logs of users, the question of how to define the threshold number of calls for creating edges can be a tricky one, which brings us significant noise. If we then directly conduct downstream tasks, such as vertex classification or link prediction based on noisy networks, this will further deteriorate the downstream task performance [40]. Nevertheless, many studies continue to draw potentially misleading conclusions from flawed network data. Inaccurate representation of networks poses a serious challenge to network analysis; accordingly it is necessary to obtain a clean network from the noisy network. Generally speaking, the noise in the graph structure can be caused by noisy links and noisy nodes; we aim to solve the noisy link problem in this paper.

Given a noisy network, in this paper, we study a novel method of denoising the network that provides better representation of the underlying topological information, thus contributes to downstream applications. In practice, the vertex set of a network is easier to determine and contains less noise than the edge set. Moreover, if a vertex is an anomalous one, it can be isolated by removing all the edges connected with it. We therefore let the denoised network has the same vertices as the original network, but with a new set of edges. The denoised network is expected to 1) preserve and even better represent the underlying topological information of the original network; and 2) contribute to downstream applications. However, it is nontrivial and remains several challenges.

A straight-forward idea is to formulate the network denoising problem as link prediction, however, it ignores the complex non-local, long-term dependency between the creations of edges. Take an example to illustrate the necessity of this dependence in Figure 1(a), where a noisy network is given. The creations of the missing edges (B, C) and (B, D) depend on each other: the existence of edge (B, D) will provide one more common neighbor to B and C, which increases the likelihood of edge (B, C). Network properties such as connectivity or scale-free property [27] ensure that the long-term dependency will exist even for those edges that are separated far away from each other. How to capture and model the non-local, long-term dependency between the

- *J. Xu and J. Lu are with the State Key Laboratory of Industrial Control Technology, College of Control Science and Engineering, Zhejiang University, Hangzhou, 310027, China, and are also with Zhejiang Laboratory, Hangzhou, 311121, China. E-mail: {xujr, lujg}@zju.edu.cn.*
- *Y. Yang is with the College of Computer Science, Zhejiang Laboratory, Hangzhou, 311121, China. E-mail: yangya@zju.edu.cn.*
- *S. Pu, Y. Fu and W. Jiang are with Hikvision Research Institute, Hangzhou, 310051, China. E-mail: {pushiliang, fuyao, jiangweihao5}@hikvision.com.*
- *J. Feng is with State Grid Corporation of China, Hangzhou, 310007, China. E-mail: junefeng.81@gmail.com.*
- *C. Wang is with FinVolution Group, Shanghai, 201203, China. E-mail: wangchunping02@xinye.com.*

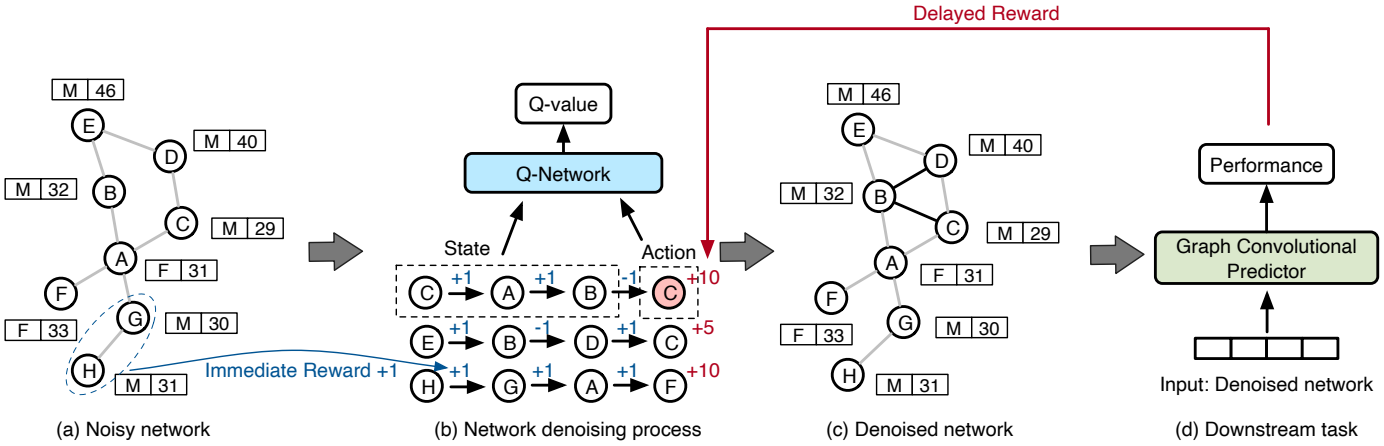


Fig. 1: Illustration of the network denoising problem and our deep reinforcement learning framework. Fig. 1(a) shows a noisy network where each vertex indicates a user and edges represent social relations (e.g., friendship) between individuals. We have two features for each user, namely sex (F/M) and age respectively. In the network, there are two missing edges (B, C) and (B, D), and an incorrect edge (G, H). Fig. 1(b) presents the proposed network denoising process via a deep reinforcement learning framework, where multiple edge sequences are generated and form a clean network, as shown in Fig. 1(c). In Fig. 1(d), the denoised network is then employed to carry out a downstream task such as vertex classification, the performance of which will in turn influence the network denoising process.

existences of edges is the first challenge in this task.

Secondly, another example in Figure 1(a): H and G share similar personal features, the incorrect edge (H, G) is difficult to detect. Taking recommendation system as our downstream application, intuitively, we will create the edge if it brings the correct information to reveal the similar product interests of H and G, and further improves the recommendation performance. Conversely, edges that decrease the performance are more likely to be noisy. This idea leads us to consider utilizing the feedback from downstream applications to guide the denoising process. However, the feedback from the downstream application can be obtained only when we have obtained the denoised network, which is typically delayed and leads to the second challenge.

The third challenge is caused by the lack of supervised information available for noisy edges, which limits the feasibility of traditional machine learning technologies.

To deal with the above challenges, in this paper, we propose a novel model to denoise networks. We name our model *NetRL*, i.e., *Net*work Denoising via *R*einforcement *L*earning. To be specific, we formulate the problem of network denoising as edge sequence generation over the network, i.e., learning to generate sequences of edges that are plausible in a clean network, which can be further formulated as a Markov Decision Process [2]. We then put the edge generation into a reinforcement learning framework [24]. NetRL plays the role of an RL agent that operates within a task-aware network denoising environment. It selects the best action of the next vertex to be connected, and is trained via Double DQN [30] to optimize reward. A new denoised network is successively constructed by connected vertices. Figure 1 shows the general structure of our framework. It illustrates how the network denoising process and the downstream task interact with each other and evolve together.

There are several advantages to our approach: Firstly, instead of directly generating edges by computing the similarity of pairwise vertices or based on domain knowledge, our RL agent will take the best action conditioned on the edges that have been generated so far, thus is capable of handling long-term, non-local dependency. Secondly, our method is task-driven which naturally utilizes the information provided by downstream tasks. Meanwhile, the reinforcement learning method naturally provides a way to utilize the feedback from the downstream application by formulating a delayed reward. Thirdly, the denoising process can be learned through trial-and-error search, attempting to produce a strong preference for a more optimized path and obtain feedback from rewards or punishments on a given noisy network and downstream task, and therefore does not require supervised information about which edges are noisy.

We summarize our contributions in this paper as follows:

- We formulate the problem of network denoising from the original noisy network to improve its quality and further optimize the downstream task.
- We propose a novel and expressive method, *NetRL*, which adopts a deep reinforcement learning framework for network denoising and allows direct optimization of task-specific objectives. We further propose a *Diversified Experience Replay* mechanism to ensure the RL agent can learn efficiently from more diverse transitions and take fully advantage of the delayed reward.
- Extensive experimental results show that NetRL is widely applicable for denoising multiple types of networks by exhibiting competitive results on downstream tasks, while still preserving important properties of the original network. We also present a case of extending NetRL to the network generation scenario, which aims to generate all edges only according to vertex features.

2 Problem Definition

Before introducing our method, we provide the necessary definitions and formulate the problem in this section. The

TABLE 1: Description of some major notations

Notation	Description
G, G^*	The input noisy network and the denoised network
V, v_i	The set of vertex and the vertex i
E, E^*, e_{ij}	The edge set of input noisy network, the edge set of denoised network and the edge (i, j)
X, $\mathbf{x_i}$, x_{im}	The feature matrix, the feature vertor of vertex v_i and the m-th feature of vertex v_i
A, D	The adjacency matrix and the diagonal degree matrix
S, l, l_i	The edge sequence set, an edge sequence and the i-th vertex in an edge sequence
f_i	The topological representation of vertex v_i in the input network
p_t	The representation of edge sequence at time t
s_t, a_t, r_t	The state, the action and the reward at time t
$\mathcal{D}$	The reply memory
ρ_t	The changeable capacity ratio of Diversified Experience Reply
$\|S\|$	The maximum number of edge sequences
N	The number of vertices
M	The number of features of vertex
F	The topological embedding size of vertex
T	The time window length
$\|L\|$	The maximum length of an edge sequence
K	The action candidate pool size
C	The target network update period
c	The trade-off between the immediate reward and the delay reward
β	The trade-off between r_t^{io} and r_t^{ih}
B, b	The number of hidden layers of GCN and the b-th hdden layer of GCN
y_i	The label of vertex v_i
Y, $\mathcal{Y}_l$	The training labels and the vertex index in the training set

major notations used in our proposed model can be found in Table 1.

Formally, let $G = (V, X, E)$ be an undirected network we aim to study in this paper, where $V = \{v_1, \ldots, v_N\}$ denotes the set of vertices in the network (with $|V| = N$), $X^{N\times M}$ representing the feature matrix, and $E = \{e_{ij}|v_i, v_j \in V\}$ indicates the set of edges. Each vertex is assigned M features, where each element $x_{im} \in X$ indicates the m-th feature of v_i.

In practice, we often obtain G by sampling from a complete network or by following certain rules. Either, it is inevitable that G will contain noisy information, especially at the edge level, as it is easier to give a certain vertex set in most cases. In this paper, we focus on studying networks with noise for edge-existence and leave vertex-level denoising as future work. That is, given G, there are some *incorrect edges* that should not exist in E, while a few meaningful and correct edges are missing in E. Thus, it is necessary to study how to denoise the original network G, which can be formally formulated as the problem of network denoising in the following way:

Definition 1. *Network Denoising. Given a noisy network $G = (V, X, E)$, the goal of network denoising is to construct a clean network $G^* = (V, X, E^*)$ by removing incorrect edges from E and creating meaningful edges for E^*. The denoised network G^* is expected to provide a better representation of the connections between vertices than the original network G. Meanwhile, G^* can contribute to the downstream applications.*

In this paper, we take vertex classification as the example of downstream application. Our goal is to feed the denoised network to a classifier (GCN as an example) , and obtain a better performance comparing with that based on the original network.

3 Our Approach

3.1 General Description

We first introduce our general idea and the overall structure of our method. Given a noisy network $G = (V, X, E)$, we generate a set of new edges E^* that forms a clean network $G^* = (V, X, E^*)$. To effectively generate E^*, we propose a deep reinforcement learning framework, where an RL agent takes the duty of creating edges.

In particular, our agent starts from a randomly selected vertex v_i, and constructs a sequence $l = (v_i)$. At the next time, the agent takes an *action* to select a vertex v_j to be connected to v_i, then an edge is created between v_i and v_j, and v_j is added to the sequence, i.e., $l = (v_i, v_j)$. We call l as *generated edge sequence* (edge sequence for short) as the agent has created edges between adjacent elements (vertices) in l. Notice that by this way, l is actually a path in our denoised network G^*. It stops extending the current l and start to generate another edge sequence when the length of l reaches $|L|$ or a closure is formed (i.e., a vertex has been selected twice). The agent finishes the whole edge generation process when the total number of edges satisfies our setting. We define $S = \{l^1, \cdots, l^{|S|}\}$ as the set of edge sequences. In turn, for the denoised network, we add the edge e_{ij}^* to E^* iff v_i and v_j, regardless of their order, appear adjacently in at least one sequence in S.

In the above edge sequence generation process, we start the edge sequence from a randomly selected vertex. The reasons are as follows. First, in the initial stage of reinforcement learning, the RL algorithm has not learned well, thus an edge sequence is more like a random walk on the graph. Considering that a random walk converges to a stationary distribution,which is independent of the starting vertex, we directly choose a random vertex to start. Second, in order to have more chance for RL agent to explore

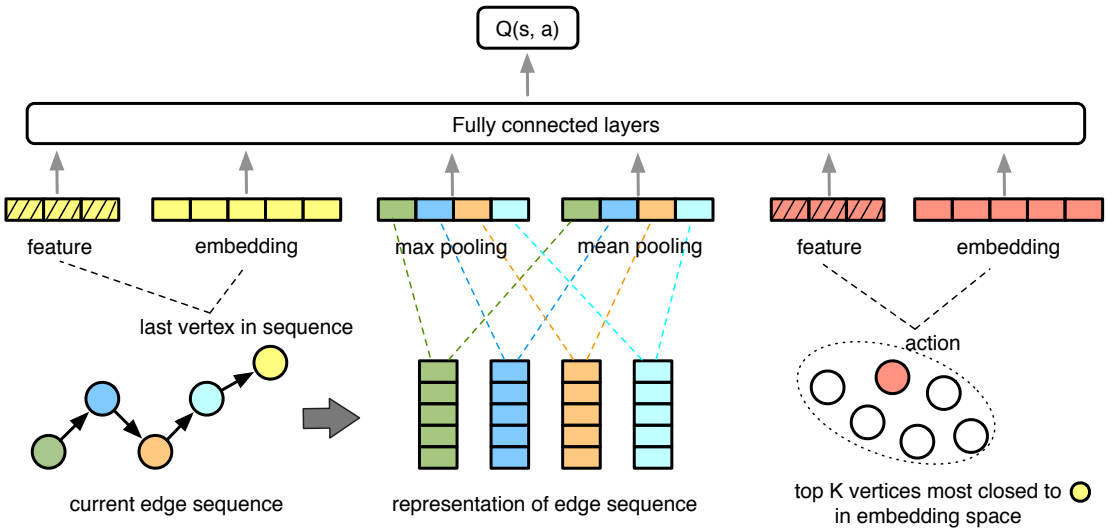


Fig. 2: The structure of the Q-network in our framework. The representation of *state* is composed of *feature*, *embedding* and *edge sequence*, while the representation of *action* is composed of *feature* and *embedding*. To be specific, *feature* captures node attributes, and *embedding* captures network properties, which varies with different network in practice.

when the RL algorithm has learned very well, we choose a randomly selected vertex to start, hopefully covering more global structure and leading to long-term benefit.

To illustrate the detailed design of our RL agent, we next introduce the state, action and reward representation.

- **State.** The state s_t in our framework is a general description of the edges that have been generated so far. It is composed of the features and topological information of the current vertex, along with the topological information of every vertex in the current edge sequence l. For a particular vertex v_i, its vertex feature (e.g., age, gender, occupation, etc.) is given by $\mathbf{x_i}$; we also use network embedding technologies such as DeepWalk [7] to obtain a representation $f_i \in \mathbb{R}^{1\times F}$ of v_i's topological information. Notice that if vertex features can not be obtained, we will only use the topological information as state representation. It is also worth to mention that f_i contains noisy information to a certain extent. However, in practice, the overall structure of the noisy network is still reliable. Therefore, we believe that f_i can still help in our task. See detailed formulation of s_t in Section 3.2.
- **Action.** Action a_t is defined to indicate which vertex v is to be added to the current edge sequence l. After taking an action, the edge sequence l will be updated by (l, v).
- **Reward.** Reward r_t is used as feedback to guide the whole denoising process. It is composed of two parts: reward from original network and reward from downstream application. We call the former the immediate reward, as it can be captured immediately, while the latter is termed the delayed reward as the downstream application can only be conducted after completing the network denoising process. These two rewards together ensure that our denoised network will preserve and even better represent the topological information of the original network, and further benefit the downstream application. See details in Section 3.3.

3.2 Q-Network Structure

Given the current state s_t, the agent aims to find the action a_t (which vertex to add) that may lead to the maximum reward in the long run, following the Bellman Equation [24]. It is thus essential to estimate the long-run reward based on a particular state-action pair. To do this, we employ a Q-network to approximate the action-value function $Q(s_t, a_t)$.

In general, we construct a deep neural network as the Q-network. Figure 2 presents the overall structure of our deep Q-network, which takes state s_t and action a_t as input to learn the mapping from the generated edges to potential rewards. One challenge here is that, since the vertices in the network are numerous, the state space and the action space are extremely large. To handle this issue, we encode state and action into a continuous low-dimensional space.

We represent the state s_t at time t as a continuous real-valued vector, which encodes the following information: 1) The rich information of the current vertex, including the vertex's topological representation and features; 2) The sequential information from the generated edge sequence. Since the generated edge sequence is very long in most cases, directly concatenating the information of each vertex in the sequence would cost many computation resources. We thus utilize pooling to obtain a summarized version of all vertices in the generated edge sequence. Considering that both the smoothed information (e.g., the edge sequence forms a circle) and the sharp information (e.g., a certain vertex is a hub vertex) of vertices in the sequence are important, we utilize both mean pooling and max pooling to summarize the generated edge sequence. We first use a vector p_t to define the representation of an edge sequence $(v_{l_{t-T}}, \cdots, v_{l_t})$, where l_i indicates the index of the i-th vertex in the edge sequence as

$$p_t = \begin{pmatrix} \text{max-pooling}\left\{f_{l_{t-T}}, \cdots, f_{l_t}\right\}, \\ \text{mean-pooling}\left\{f_{l_{t-T}}, \cdots, f_{l_t}\right\} \end{pmatrix} \tag{1}$$

where the time window length T denotes the maximal view range that can be observed by the RL agent, and f_i is the

topological representation vector of v_i, obtained by some network embedding algorithm. We conduct max-pooling and mean-pooling on the representation vectors of the current edge sequence $\{f_{l_{t-T}}, \ldots, f_{l_t}\} \in \mathbb{R}^{T \times F}$ to reduce the spatial dimension and simplify the model complexity.

Intuitively and empirically, the last vertex v_{l_t} in the edge sequence will be more important than the others, as it directly influences the next vertex to be added into the edge sequence. Therefore, we encode the state s_t as the vector, which is concatenated by the feature vector x_{l_t} of v_{l_t}, the topological representation vector f_{l_t} of v_{l_t}, and the representation of other vertices in the edge sequence (i.e., p_{t-1}), as follows:

$$s_t = [x_{l_t}; f_{l_t}; p_{t-1}] \tag{2}$$

At the beginning ($t = 0$), the agent starts with an edge sequence with only one randomly selected vertex.

As for the action a_t, by assuming the selected vertex is v_i, we use the feature vector x_i and the topological representation vector f_i of vertex v_i to encode a_t as

$$a_t = [x_i; f_i] \tag{3}$$

In addition, to reduce computational complexity and improve model efficiency, we construct an action candidate pool for each vertex v_j. When v_j becomes the last vertex in the current edge sequence, we only consider adding the top K nearest vertices to v_j in the topological embedding space. In other words, v_i will be considered to be added after v_j only when their topological representation vectors, i.e., f_i and f_j, are close enough. It is easy to see that when making an action, the computational complexity would reduce from $O(N)$ to $O(K)$ ($K \ll N$). This special design can also be interpreted as a certain constraint that makes the newly created edges not too far from the original network.

The Q-network maps the vector representation of the current vertex, the vertices traversed before, and the next vertex to be reached as inputs to the neural network shown in Figure 2, which outputs a single neuron approximating $Q(s_t, a_t)$.

$$Q(s_t, a_t) = f_\theta([s_t; a_t]) \tag{4}$$

where $[;]$ denotes vector concatenation, and $f_\theta(\cdot)$ is a fully connected neural network with model parameter θ.

3.3 Reward Computation

We define two types of rewards: immediate reward r_t^{immed} and delayed reward r_t^{delay}, to indicate the utility of the denoised network and guide the training process of the Q-network. During the training process, we define a *terminal state* in our framework, which will be triggered when the agent goes through a given length of decision sequence. Therefore, the agent will receive a delayed reward at the terminal state, and will receive an immediate reward at other states. We then introduce our elaborate design for these two rewards.

Immediate reward. The inspirations for immediate reward are two fold: 1) the denoised network should preserve the topological information of the original network; 2) as network homophily suggests, vertices with similar attributes are more likely to be connected. Therefore, we define the immediate reward r_t^{immed} composed of two parts: r_t^{io}, obtained from the first motivation, and r_t^{ih}, calculated according to the second motivation.

$$r_t^{immed} = \beta * r_t^{io} + (1 - \beta) * r_t^{ih} \tag{5}$$

where hyper-parameter β regulates a trade-off between r_t^{io} and r_t^{ih}.

For the first term, r_t^{io} is computed to capture the major topological structure of the original network. Assuming that the last vertex of the current edge sequence l is v_i, and the next vertex to be added into the edge sequence is v_j, we give a positive reward if the newly created edge e_{ij} exists in the original network. Moreover, to avoid creating duplicated edges in the edge sequence, we do not give the positive reward if the vertex pair (v_i, v_j) or (v_j, v_i) exists in the current edge sequence. Putting things together, r_t^{io} is defined as:

$$r_t^{io} = \begin{cases} -1, e_{ij} \notin E \vee (v_i, v_j) \in l \vee (v_j, v_i) \in l \\ 1, \textit{otherwise} \end{cases} \tag{6}$$

For the second term, vertex preference is a phenomenon of homophily in network science, which speeds up the information diffusion process and is useful for improving network quality. In our work, we use vertex labels to represent this kind of information. To be specific, we have:

$$r_t^{ih} = \begin{cases} 0, y_i = ? \vee y_j = ? \\ -1, y_i \neq y_j \wedge y_i \neq ? \wedge y_j \neq ? \\ 1, \textit{otherwise} \end{cases} \tag{7}$$

where y_i denotes the label of v_i, $y_i = ?$ if the label is missing in the training data. Here, we assume that 20% labels are missing in all datasets to simulate the actual condition.

Delayed reward. The delayed reward r_t^{delay} indicates how the denoised network contributes to the downstream application, which is estimated by a classifier or task-specific machine learning model. As an example, we utilize GCN as the classifier on the vertex classification task for illustration (when conducting other tasks, we can apply other task-specific models in a similar way). Formally, we define

$$\begin{aligned} r_t^{delay} = [&\textit{score}(H_t^{(B)}(A_t^*, X), Y) - \\ &\textit{score}(H_t^{(B)}(A^{ori}, X), Y)] * c \end{aligned} \tag{8}$$

$$H_t^{(b+1)}(A, X) = \phi\left(\tilde{D}_t^{-\frac{1}{2}} \tilde{A} \tilde{D}_t^{-\frac{1}{2}} H_t^{(b)} W_t^{(b)}\right) \tag{9}$$

where A_t^* is the adjacency matrix of the denoised network; A^{ori} is the adjacency matrix of the original network; X is the vertex feature matrix; Y is the training labels; $H_t^{(b)}$ is the b-th hidden layer of a deep classifier (see details later), which can be implemented by many ways; $H_t^{(B)}$ is thus the predicted label; *score* is the evaluation measure (f1 score in our setting), indicating how well the predicted labels fit the ground truth; c is a constant, in order to change the magnitude of task-specific feedback, which can be treated as a trade-off between delayed reward and immediate reward.

In this paper, we take Graph Convolutional Network (GCN) [25] as an example of our classifier H_t, a state-of-the-art technique that can achieve high performance in

Algorithm 1 The NetRL algorithm

Input: Episode number *episodes*. The maximal length of edge sequence $|L|$. Target network update period C. The greedy coefficient ε. Learning rate α. Q-network with parameters θ.

Initialize current step number: $step = 0$
Initialize the target network: $\theta^- = \theta$
repeat
  Randomly select a vertex v_{l_0};
  Initialize state $s_t = (x_{l_0}, f_{l_0}, p_0)$;
  for $t = 0$ **to** $|L|$ **do**
    With probability ε select a random action a_t;
    Otherwise select $a_t = argmax_a Q^*(s_t, a; \theta)$;
    Execute action a_t in emulator and observe immediate reward r_t^{immed} according to Eq. 5 and next state s_{t+1};
    Set $terminal = 0$;
    Store transition $\left(s_t, a_t, r_t^{immed}, s_{t+1}, terminal\right)$ in $\mathcal{D}_{immed}$;
    if $t == |L|$ **then**
      Reconstruct denoised network A_t^* via L.
      Compute delayed reward r_t^{delay} according to Eq. 8 by GCN;
      Set $terminal = 1$;
      Store transition $\left(s_t, a_t, r_t^{delay}, s_{t+1}, terminal\right)$ in $\mathcal{D}_{delay}$;
    end if
    $step = step + 1$
    Sample minibatch of transitions $(s_j, a_j, r_j, s_{j+1}, terminal)$ from $\mathcal{D}_{immed}$ and $\mathcal{D}_{delay}$ with capacity ratio ρ_t according to Eq. 12;
    $\mathscr{Y}_j = r_j + (1 - terminal) * \gamma Q(s_{j+1}, argmax_{a'} Q(s_{j+1}, a'; \theta^-))$;
    Update $\theta = \theta - \alpha \cdot \nabla_\theta \left(\mathscr{Y}_j - Q(s_j, a_j; \theta)\right)^2$;
    if $step \pmod C == 0$ **then**
      Set $\theta^- = \theta$;
    end if
  end for
until *episodes* finish

vertex classification. In particular, GCN directly encodes the network structure using a neural network, builds the mapping between the given network and vertex labels, and is trained in a supervised way based on a set of training labels. In Eq. 9, $\tilde{A} = A + I$ is the adjacency matrix and I is the identity matrix; $\tilde{D}_{tii} = \sum_j \tilde{A}_{ij}$; $W_t^{(b)}$ is a trainable weight matrix of the b^{th} layer. $\phi(\cdot)$ denotes the activation function. $H_t^{(b)}$ is the representation of the b-th layer, and we define that $H_t^0 = X$. Here, we apply a B layers GCN to $A \bigcup X$ to predict the final vertex labels $H_t^{(B)}(A, X)$.

3.4 Model Learning

Learning process. In our method, the GCN classifier provides a mechanism for computing the delayed reward of the denoised network. We aim to maximize the expected total reward, i.e, minimize the cross-entropy error described below:

$$\mathcal{L}_{GCN} = -\sum_{i \in \mathcal{Y}_l} Y_i \ln H_{ti}^{(B)}, \tag{10}$$

where $\mathcal{Y}_l$ is the set of node indexes in training set, Y_i and $H_{ti}^{(B)}$ are the true label and the predicted label of v_i respectively. Here, we apply gradient descent to optimize the GCN algorithm.

Once we obtain a high-value delayed reward, the agent will learn more from it. Meanwhile, once the agent chooses high-quality actions according to $Q(s, a)$, this will also lead to better performance in GCN. We apply Double DQN [30] to train our RL framework (In earlier experiments we also tried some policy optimization methods, but found Double DQN works more stable). We hope that the Q-network can fit the total reward well in order to estimate the utility of each decision, which raises the problem of minimizing the mean-squared error in Q-values:

$$\mathcal{L}_{RL}(\theta) = \mathop{\mathbb{E}}_{(s,a,r,s') \sim \mathcal{D}} [(r + \gamma Q(s_{j+1}, argmax_{a'} Q(s_{j+1}, a'; \theta^-)) - Q(s, a; \theta))^2] \tag{11}$$

where γ is the discount factor. Here, we apply adaptive moment estimation (Adam) to optimize this objective. In order to maintain a stable update, we take advantage of a target Q-network, parameterized as θ^-, and an original Q-network, parameterized as θ. The parameters in the target Q-network are updated far more slowly than the original ones, which can only be replaced $\theta^- \rightarrow \theta$ periodically.

Diversified experience replay. Considering the sparsity of delayed reward and the training efficiency of our model, we are motivated to improve the experience replay mechanism. Experience replay [29] proposed in DQN [26] enables Q-networks update parameters through the recent experiences stored in a replay memory, which in turn stabilize the training process. However, there are two issues to be solved: 1) When the training process is relatively stable, there are always some specific state-action pairs with large Q-values. As a result, many repeated and redundant experiences are stored in replay memory. The transitions sampled from the replay memory are lack of diversity, which will make the training process less efficient, and more likely to be trapped in local optimum. For example, some edges with large Q-values are more likely to be repeatedly generated, which may lead to a sparse denoised network with few edges; or, some vertices will always be selected, which may lead to a large-degree vertex. 2) The delayed reward is much sparser than the immediate reward. Thus, we divide the whole replay memory $\mathcal{D}$ into two replay memories $\mathcal{D}_{immed}$ and $\mathcal{D}_{delay}$ with different capacities. However, the ratio between $\mathcal{D}_{immed}$ and $\mathcal{D}_{delay}$ is hard to define. In the early training stage, transitions sampled from $\mathcal{D}_{delay}$ is redundant due to its sparsity, learning too much from the delayed reward will lead to unstable and inefficient training process. In the later training stage, learning too little from the delayed reward will make the RL agent short-sighted.

Here, we propose a new *Diversified Experience Replay* mechanism to ensure the RL agent learn efficiently from more diverse transitions and take fully advantage of the delayed reward. The mechanism of diversified experience transitions enhances the ability of global convergence and also greatly reduces training time. To deal with the first issue, we first extract the unrepeated transitions, and then conduct uniform sampling among them; if the number of unrepeated samples do not meet the required sample size, the rest will be sampled from the replay memory. To deal

with the second issue, we define a changeable capacity ratio ρ_t between $\mathcal{D}_{immed}$ and $\mathcal{D}_{delay}$ at time t.

$$\rho_t = max\left(\rho_{min}, \zeta^{step/s_d}\rho_0\right) \tag{12}$$

where ρ_{min} constrains the minimal value of ρ; ρ_0 is the initial value of ρ, which is set to a relatively large value as immediate reward is more important in early stage of training; $\zeta \in (0, 1)$ is the decay rate; *step* is the current step number; and s_d is the decay step.

The proposed NetRL algorithm is presented in Algorithm 1. In the NetRL algorithm, the time complexity mainly comes from line 8, 14 and 20. According to [36], a basic GCN takes $O(|V|\,M) \simeq O(|V|)$ (line 14) as we have $M \ll |V|$ or a sparse feature matrix X in practice (Note that for message passing-based GCNs, the update equation only depends on the neighborhood of a specific node, and independent of graph size, making the complexity $O(|E|)$ reducing to $O(|V|)$ for sparse graphs). When applying MLP to compute the Q-value of each (state, action) pair for $K\,|L|$ times in each episode , it would take $O\left(K\,|L|\,(M + F + T)\right) \simeq O\left(K\,|L|\right)$ (line 8 and line 20) as we have $(M + F + T) \ll K\,|L|$ in practice. Thus, the overall time complexity of NetRL algorithm is $O(K\,|L| + |V|)$.

4 Experiments

4.1 Experimental Setup

In the experiments, we aim to conduct two downstream tasks, i.e., vertex classification and social tie prediction task. Vertex classification is the most common task, and its goal is to determine the label of a vertex. The problem of social tie prediction can be formulated below. Given a social network and a particular edge in the network, we aim to determine if the edge is a strong tie or a weak tie.

Datasets. We use seven different networks, which can be categorized into:

- *Mobile network.* We aim to conduct two practical tasks (i.e., fraud detection and social tie prediction) on mobile networks. Two real-world mobile networks, i.e., FinV and Telecom, are utilized.
  1) FinV is a dataset provided by FinVolution Group [1] [41]. If two registered users have made a phone call, we create a link between them. For each user, we obtain their demographic information (including age, sex, birthplace, educational level, marital status, work type and so on) as user attributes. For each phone call log, we can obtain its starting time and its ending time. We fortunately have some loan records and accordingly define a user who has 90 days overdue repayment as a fraudster. We label 2,140 fraudsters among 11,410 users who have at least one loan history record. We also have the call duration of each edge and accordingly define a call lasts greater than 1 minutes as a strong tie, otherwise it is a weak tie, thus 4,151 strong ties are identified.
  2) Telecom is a dataset provided by China Telecom [2], which is formed in a similar way as FinV. We label a user as the fraudster if he/she was reported as a fraudster by someone either to Baidu [3] or Qihoo 360 [4]. These ground truth data comes from a large number of complaints and therefore has a very high confidence. In this way, we obtain 1,4361 fraudsters among all the users. Besides, among 1,575,498 edges, we identify 605,203 strong ties whose call duration is greater than 1 minutes.
  Notably, these two datasets contain noise to some extent. For example the links built upon the calls misdialed are most likely to be noisy links, as they do not represent the actual "social" relationships.
- *Social network.* BlogCatalog and Flickr, two benchmark social networks. Vertices represent users and edges represent social relations. All the vertices are divided by user interests into different categories. The vertices of BlogCatalog can be classified into 6 categories and that of Flickr fall into 9 categories. Regarding the node features, we conduct SVD decomposition on them to reduce dimension as [32] does. Because these two datasets are extracted from social media, they contain noise to some extent .
- *Academic*. Cora and Citeseer, two benchmark citation networks. Vertices are documents and edges are citation links between documents. Each vertex has a human-annotated topic as a class label as well as a feature vector, which is a sparse bag-of-words representation of the document. The vertices of Cora fall into 7 classes and that of Citeseer are divided into 6 categories. Since the feature dimension of Citeseer is too large, which may slow down the training process. Hence, we construct a TFIDF matrix for Citeseer and conduct SVD decomposition to reduce its dimension as [32] does.
- *Web.* Wiki, a web network. Each Wikipedia document represents a vertex and the connection between them forms an edge. Each vertex has substantial text information as its features in the form of TFIDF and are divided into 6 categories. We remove the documents whose category ratio is less than 5% to balance the dataset. We also conduct SVD decomposition on node features to reduce dimension as [32] does.

Dataset statistics are summarized in Table 2.

TABLE 2: Dataset statistics

Dataset	Type	#Vertices	#Edges	#Features
FinV	Mobile network	11,410	12,262	102
Telecom	Mobile network	290,483	1,575,498	261
BlogCatalog	Social network	5,196	171,743	8189
Flickr	Social network	7,575	239,738	12,047
Cora	Citation network	2,708	5,429	1,433
Citeseer	Citation network	3,327	4,732	3,703
Wiki	Web network	1,803	8,915	4,973

Baselines. When conducting the vertex classification task, we compare our approach with the following baselines: 1) *Label Propagation (LP) [3].* ParWalks [3], a label propagation method via partially absorbing random walk, is applied. 2) *GCN [25].* Semi-supervised node classification with GCN. GCN and LP are conducted on the original network, which

1. FinVolution Group, the first and one of the online lending platforms in China.

2. China Telecom, one of the world's largest providers of integrated telecommunication services.

3. http://www.baidu.com, one of the largest Internet companies in the world.

4. http://www.360.com, a Chinese internet security company.

TABLE 3: Experimental results on vertex classification, where precision, recall and f1 for the binary classification is given on FinV and Telecom, and macro precision, macro recall and macro f1 for multi-class classification is given on other datasets.

	FinV			Telecom			BlogCatalog			Flickr		
	Prec.	Rec.	F1.	Prec.	Rec.	F1.	Prec.	Rec.	F1.	Prec.	Rec.	F1.
LP	0.1432	0.1345	0.1274	0.4950	0.4988	0.4926	0.4334	0.5202	0.4009	0.3473	0.3754	0.3327
GCN	0.1848	0.1879	0.1861	0.8587	0.5921	0.6390	0.7251	0.6934	0.6885	0.5632	0.5033	0.4746
NetGAN	0.1548	0.1432	0.1442	-	-	-	0.2546	0.2264	0.1444	0.2718	0.2387	0.2100
GraphGAN	0.1909	0.1160	0.1496	-	-	-	0.6403	0.4694	0.4678	0.6034	0.4856	0.4917
NE	0.2149	0.1837	0.1993	-	-	-	0.7333	0.7305	0.7303	0.4346	0.4383	0.4318
NetD	0.1920	0.1701	0.1804	0.8450	0.5400	0.5612	0.7234	0.7063	0.6976	0.5580	0.4980	0.4691
NetRL	**0.3932**	**0.2541**	**0.3087**	**0.9004**	**0.7041**	**0.7669**	**0.8410**	**0.8409**	**0.8400**	**0.5649**	**0.5750**	**0.5620**

do not denoise at all. 3) *NetGAN [13].* A graph generation model, also constructs networks through random walks. We conduct GCN [25] on the constructed network by NetGAN; 4) *GraphGAN [33].* The generator in this work can be treated as a graph generation model, which fits the underlying true connectivity distribution. We conduct GCN [25] on the generated network by GraphGAN; 5) *NE [31].* The most related work in network denoising, which optimizes the adjacency matrix end-to-end. We conduct GCN [25] on the denoised network by NE. 6) *NetD [37].* A network denoising method, by which the denoised network is generated by removing unimportant connections as noise. We then perform GCN [25] on the denoised network by NetD.

When conducting the social tie prediction task, the classifier GCN [25] can not handle link-level tasks well, thus we utilize the variational graph auto-encoder (GAE) [11], a model that can achieve good performance in link-level tasks, as the running example of our downstream task classifier.

Implementation Details. Here, we provide detailed implementation details of our experiments. In the training phase, we set the maximal length of each edge sequence $|L|$ as 100 when generating the edge sequences. The time window length T considered in Eq. 1 is set as 10 when constructing Q-Network. We choose the embedding size of DeepWalk F as 64, the action candidate pool size K as 100. When computing the reward, the immediate reward parameter β that regulates a trade-off between r_t^{in} and r_t^{ih} is set to be 0.7 and the constant c that controls the magnitude of delayed reward is set to be the same value as $|L|$. The replay memory size of the delayed reward is set to be 500. Regarding the replay memory size of the delayed reward, the minimal value ρ_{min}, the initial value ρ_0, the decay rate ζ and the decay step s_d of the changeable capacity ratio are 100, 150, 0.3 and 2, respectively. The discount factor γ in Objective 11 is 0.5, the target network update period C is 4, and the greedy coefficient ε decays according to $\varepsilon_{t+1} = \frac{\varepsilon_t}{t^\mu}, \varepsilon_0 = 1$, where μ denotes its decay rate that is set to 3e-6. The feature dimension obtained from SVD composition for Citeseer and Wiki is set to be 200. For GCN and GAE, the parameters are set to their default values. For the Q-network, we use a one layer MLP, and ReLU for the activation function of hidden layers, where the weights are initialized by Xavier. Pooling operations reduce the dimension of edge sequence into 4. During training the Q-network, we use a batch size of 64 and train it using the Adam optimizer. The learning rate α is set to be $1e$-4. In the inference phase, we generate an edge sequence for each node (i.e., $|S| = N$) and set the maximal length of each edge sequence $|L|$ as 3. Other parameters are the same as that in the training phase.

Our model is implemented in Tensorflow version 1.14.0 with CUDA version 9.0 and Python 3.5.2. All the experiments are conducted on a single machine with an Intel Xeon E5 (252GB memory) and a NVIDIA TITAN GPU (12GB memory). When conducting the vertex classification task, we use 10% of all the labels as train data and the remaining 90% for testing. When conducting the social tie prediction task, we allocate 90% of all the labels for training and 10% for testing. The reported results are averaged over 10 runs with random dataset split and random weight initializations. Our implementation is available at: https://github.com/galina0217/NetRL.

4.2 Comparison Results

Table 3 lists vertex classification performance on several noisy networks (i.e., FinV, Telecom, BlogCatalog and Flickr), and we can see that NetRL performs consistently best on different datasets (+26.1% in terms of f1). Compared with LP and GCN, ours performance gain is correlated with the help of the network denoising process. Compared with two generation models NetGAN and GraphGAN, ours is able to capture the long-term dependence between edges, while NetGAN and GraphGAN create edges without considering the edges that have already been generated. The Comparison with NE, the most related work on network denoising, suggests the utility of downstream task guidance. In addition, benefiting from the exploration mechanism of reinforcement learning, NetRL avoids the overfitting issue. Notice that the methods conducted on the generated or denoised network (i.e., NetGAN, GraphGAN, NE) sometimes perform worse than the methods conducted on the original network (i.e., LP, GCN), which suggests that it is better not to denoise if the denoised networks do not contribute to the downstream task, since network denoising is a non-trivial task.

We further evaluate our method on the social tie prediction task on FinV and Telecom dataset. Note that this task

TABLE 4: Experimental results on social tie prediction task in terms of AUC.

	DeepWalk	GAE	NetGAN	GraphGAN	NetD	NetRL
FinV	0.5212	0.5410	0.5503	0.5616	0.5517	**0.5881**
Telecom	0.5736	0.6115	0.6195	0.6304	0.6371	**0.6515**

TABLE 5: F1 score with different noise rates on vertex classification task.

	FinV					Telecom					BlogCatalog					Flickr				
Noise rate	0%	5%	10%	15%	20%	0%	5%	10%	15%	20%	0%	5%	10%	15%	20%	0%	5%	10%	15%	20%
LP	0.1274	0.1241	0.1146	0.1144	0.1027	0.4926	0.4927	0.4918	0.4905	0.4913	0.4009	0.3709	0.3465	0.3334	0.2978	0.3327	0.3192	0.3007	0.2894	0.2784
GCN	0.1861	0.1721	0.1647	0.1588	0.1310	0.6390	0.7187	0.7146	0.7020	0.6690	0.6885	0.6878	0.6492	0.6530	0.6552	0.4746	0.4381	0.4407	0.4471	0.4104
NetGAN	0.1442	0.1439	0.1084	0.1090	0.0731	-	-	-	-	-	0.1444	0.1060	0.2043	0.1331	0.0932	0.2100	0.1620	0.1444	0.1473	0.1414
GraphGAN	0.1496	0.1341	0.1450	0.1335	0.1263	-	-	-	-	-	0.4678	0.4666	0.4852	0.4576	0.4740	0.4917	0.4837	0.4852	0.4576	0.4740
NE	0.1393	0.1301	0.1201	0.1211	0.1241	-	-	-	-	-	0.7303	0.7307	0.7121	0.7068	0.6930	0.4318	0.4071	0.4315	0.4522	0.5053
NetD	0.1804	0.1710	0.1551	0.1385	0.1167	0.5612	0.5669	0.5577	0.5529	0.5393	0.6976	0.6882	0.6796	0.6703	0.6651	0.4691	0.4399	0.4044	0.3927	0.3783
NetRL	**0.3087**	**0.2956**	**0.2789**	**0.2577**	**0.2345**	**0.7669**	**0.7537**	**0.7370**	**0.7348**	**0.7461**	**0.8400**	**0.8396**	**0.8385**	**0.8248**	**0.8191**	**0.5620**	**0.5525**	**0.5665**	**0.5676**	**0.5618**

	Cora					Citeseer					Wiki				
Noise rate	0%	5%	10%	15%	20%	0%	5%	10%	15%	20%	0%	5%	10%	15%	20%
LP	0.7235	0.6845	0.6489	0.5063	0.5656	0.5260	0.5161	0.5012	0.4463	0.4094	0.1269	0.1251	0.1331	0.1247	0.1256
GCN	0.8205	0.7955	0.7811	0.7662	0.7466	0.6105	0.6001	0.5841	0.5412	0.5300	0.7217	0.7162	0.7065	0.6699	0.6392
NetGAN	0.1374	0.1465	0.1161	0.1269	0.1838	0.2379	0.2312	0.2061	0.2074	0.1904	0.2654	0.2644	0.2439	0.2186	0.1904
GraphGAN	0.3991	0.3985	0.3856	0.3733	0.3739	0.3846	0.3845	0.3816	0.3822	0.3854	0.5462	0.5449	0.5512	0.5451	0.5540
NE	0.5837	0.5844	0.5561	0.5278	0.5181	0.4515	0.4681	0.4585	0.4677	0.4935	0.6830	0.6822	0.6879	0.6819	0.6839
NetD	0.8094	0.7924	0.7749	0.7593	0.7353	0.5892	0.5803	0.5645	0.5416	0.5266	0.7301	0.7267	0.7041	0.6433	0.6236
NetRL	**0.8261**	**0.8154**	**0.7813**	**0.7661**	**0.7587**	**0.6335**	**0.6295**	**0.6166**	**0.6103**	**0.6072**	**0.7429**	**0.7349**	**0.7222**	**0.7067**	**0.6908**

requires labels on links, therefore Cora and other benchmarks without such information are not considered. GAE [11] is taken as the downstream task classifier when predicting social ties. Table 4 reports the AUC of all the comparable models except NE, because NE results in a dense matrix which can not be directly applied to the link prediction task. We can see that NetRL still outperforms other methods, which demonstrates that our method can actually benefit from different downstream tasks.

4.3 Noise Sensitivity

To evaluate the noise sensitivity of our model, we add different ratio of noise to the edge set respectively. To do this, we first randomly remove a certain proportion of all edges, and them randomly select this many unconnected vertex pairs and create new edges between these vertices (N.B. We make sure that there is no isolated nodes exist). Table 5 presents the results of vertex classification task when different ratio of noise are given. We observe that the performance of LP and GCN degrade significantly when noise becomes stronger in most cases, as these semi-supervised methods are directly conducted on the noisy networks. We can see that the two generation methods NetGAN and GraphGAN also exhibit relatively poor results, especially the performance of NetGAN, this reveals that they may not work when a noisy network is given. The performance of NE is clearly not ideal. This is likely due to the fact that NE is a denoising method only designed for strong noise environment. In contrast, our method consistently achieves comparable performance. Moreover, NetRL is much more robust to noise, showing minimal degradation in performance even noise rate increases. And even if 20% noise are introduced, the performance of NetRL is still acceptable. Overall, the results indicate that the denoised network obtained by our method does help in dealing with noise issues and results in better downstream task performance.

4.4 Preserving Structural Properties

We further conduct experiments to validate whether the denoised network encodes the structural properties of the original network. Since Cora is an relatively clean network, we take Cora as an illustrating example such that we can obtain a more clear view when comparing the clean network, noisy network and denoised network in Figure 3. We add 40% noise to Cora such that we can observe some more distinguishing patterns between clean network and noisy network. We here also provide the results of NetRL and NE when conducting the vertex classification task on Cora with 40% noise for your reference, i.e., NetRL achieves 0.6752 and NE achieves 0.5266 in terms of F1 score. Figure 3 shows the visualization by an efficient graph drawing algorithm [34] of various networks: the clean network without noise, the noisy network with 40% noise, the denoised network by NE and the denoised network by NetRL. We observe that the denoised network by NE (Figure 3(c)) does not have the capability to get an ideal denoised network when too much noise is introduced, while the denoised network by NetRL (Figure 3(d)) exhibits discernible clustering and is better at denoising complex noisy networks. The visualization further shows that the denoised network by NetRL is more sparse than the clean network and also has the natural structure similar to the clean network (Figure 3(a)). This is likely due to the clean network and the denoised network by NetRL have one thing in common: they all benefit to our downstream task.

TABLE 6: Comparison of NetRL and NE when preserving structural information of original network. We use MMD as the evaluation metric (the smaller the better).

	Cora		Citeseer		Wiki	
	Deg.	Clus.	Deg.	Clus.	Deg.	Clus.
NE	1.993	1.999	2.000	2.000	2.000	1.432
NetRL	**1.959**	**0.439**	**1.934**	**0.217**	**1.475**	**0.470**

The properties of the learned denoised networks should also be investigated quantitatively, for this purpose we provide the derived MMD scores [12] as our evaluation metrics to compare the degree distribution and the clustering coefficient distribution of the original network and the denoised network from the noisy network with 40% noise.

(a) Clean network without noise (b) Noisy network with 40% noise (c) Denoised network by NE (d) Denoised network by NetRL

Fig. 3: Visualization of Cora networks, where different colors present different vertex categories.

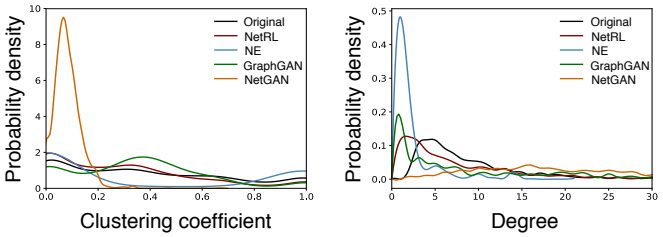


Fig. 4: Clustering coefficient and degree distributions of the original network and denoised networks of Wiki.

Table 6 shows the MMD scores (the smaller the better) of NE, which is the only work designed for denoising, and our proposed NetRL (here, we do not consider FinV, Telecom, BlogCatalog and Flickr because they are noisy themselves, which can not be taken as our ground truth, i.e., the underlying clean network). We can see that NetRL significantly outperforms NE on each dataset. For convenience of further comparison, Figure 4 visualizes the degree and clustering coefficient distribution of the network obtained by NetRL and other generation or denoising baselines, plus that of the original network. We only show the results of Wiki due to the space limit. Clearly, NetRL still performs best in capturing network properties, with the distribution closely matching the original network, which demonstrates that the intermediate representation of NetRL encodes the topology information most related to the original data. These experimental results demonstrate that NetRL can denoise networks not only benefiting the downstream task but also effectively preserving the network structural properties of the original network.

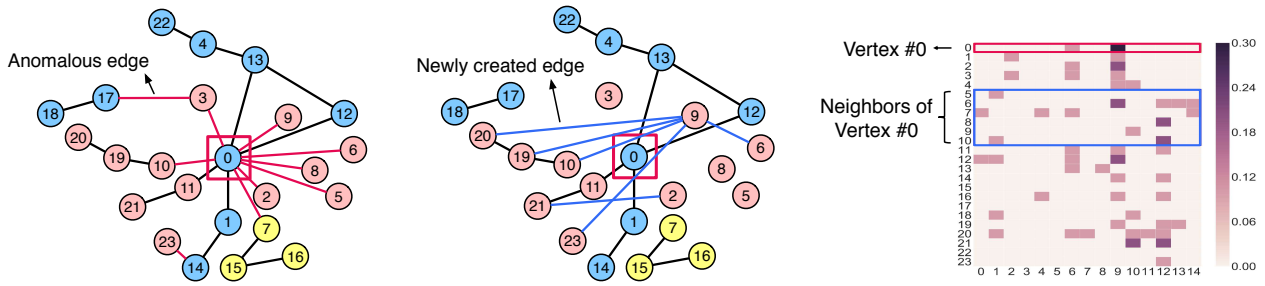
4.5 Case Study

In this section, we take a subgraph from Cora as an example to demonstrate our denoising process, shown in Figure 5. The feature vector visualization is based on 0-1 normalized features and we only show parts of important features after mean-pooling considering its high-dimension. In the original network (Figure 5(a)), vertex #0 is a high-degree vertex, while most of its neighbors possess different labels, which runs contrary to our common sense. Moreover, from feature vector visualization (Figure 5(c)), we observe that vertex #0 presents a different feature pattern compared to its neighbors that have different labels. Accordingly, we suspect that the edges (in blue) between vertex 0 and other vertices with different labels are anomalous. Taking a look at the denoised network (Figure 5(b)), it's easy to find that the anomalous edges are deleted, which is highly consistent with our suspicions. Another interesting observation is that some edges (in red) are newly created to meet network structural balance (e.g., a person's friends are more likely to be friends). Overall, our denoised network does refine the network structure and is well-suited for capturing the complex real-world networks, which helps to enhance downstream tasks.

4.6 Parameter Analysis

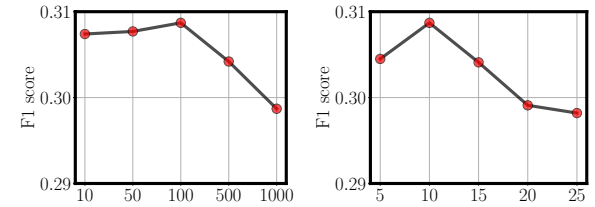
There are four important types of hyper-parameters affecting the model performance, which are the maximal length of edge sequence $|L|$, the time window T, the immediate reward parameter β that regulates a trade-off between r_t^{in} and r_t^{ih}, and the learning rate α, respectively. Here, we present the suitable range of these parameters for reference.

To determine the default parameters settings, we vary the values of $|L|$, T, β and α to observe how the performance changes. In detail, we study $|L| \in \{10, 50, 100, 500, 1000\}$, $T \in \{5, 10, 15, 20, 25\}$, $\beta \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, $\alpha \in \{1e\text{-}6, 1e\text{-}5, 1e\text{-}4, 1e\text{-}3, 1e\text{-}2\}$. It should be noted that the learning rate α remains at $1e$-4 while studying $|L|$ and β, the maximal decision sequence length $|L|$ remains at 100 while studying α and β, the time window T remains at 10 while studying α and β, β remains at 0.7 while studying $|L|$ and α. Due to space limitations, we only present the results on FinV dataset in Figure 6. From Figure 6(a), we observe that our model yields good results when $|L|$ is shorter than or equal to 100, while the performance will decrease if $|L|$ is too long. The reason is that a long edge sequence needs to be taken before we can get feedback from the delayed reward, which will significantly decrease the algorithm efficiency. One other thing to note is that it's better to set $|L|$ a bit larger when the network size is a larger one, otherwise too short edge sequence will make it hard to obtain enough information to compute the delayed reward From Figure 6(b), we can see that the best performance can be achieved when T is set to 10. It can be explained that few observed vertices in the current edge sequence can not provide us enough information, while too many observations would in turn bring noise and prevent us from capturing the most important part of edges sequence information. From Figure

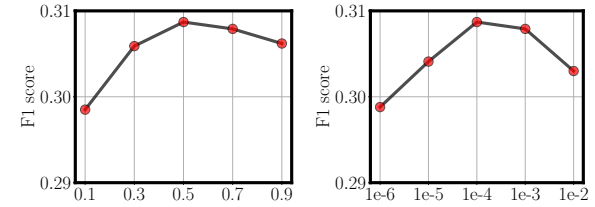


(a) Original network (b) Denoised network (c) Feature matrix

Fig. 5: Examples of a subgraph from Cora, where some representative edges and vertices are displayed and analyzed here. Vertices are assigned different colors to differentiate their labels. Red edges in original network represent the anomalous edges that we suspect, while blue edges in denoised network represent the newly-created edges.



(a) $|L|$: maximal length of edge sequence (b) T: time window



(c) β: immediate reward parameter that regulates a trade-off between r_t^{in} and r_t^{ih} (d) α: learning rate

Fig. 6: Hyper-parameter analysis

6(c), we observe that the performance decreases when a β is smaller than 0.5. Recall that NetRL with smaller β tends to consider more rewards from the original network, which will in turn introduce more noise. Thus, when we can intuitively tune this parameter to be larger if we have known there is a lot of noise introduced in the given network. Figure 6(d) shows that our model performs best when α is set to 1e-4. A bigger learning rate may lead to divergence, while a smaller learning rate may lead to slow convergence.

4.7 Extension to Network Generation

It is interesting to verify if our method can be extended for network generation, in which task we aim to construct a network only according to vertex features X and no original network is given. For the extension, we set β as 1, to ignore r_t^{io}; When defining state, we replace all the representation vectors f_i with vertex feature vectors x_i. The experimental results in Table 7 show that we can still get a good performance without the input network. Moreover, we observe that the generation performance is sometimes even better than the case that we can access the noisy network

TABLE 7: Experimental results of network generation.

	Precision	Recall	F1
FinV	0.3575	0.2088	0.2684
Telecom	0.8406	0.6394	0.7015
BlogCatalog	0.6230	0.6142	0.6120
Flickr	0.4129	0.4095	0.4057
Cora	0.7044	0.6001	0.6223
Citeseer	0.5617	0.5471	0.5462
Wiki	0.7178	0.7233	0.7158

when the noise rate is very large in FinV and Wiki. This indicates that it's better not to introduce too much original structure information when we have known that the given network is very noisy.

5 Related Work

In line with the focus of network denoising problem, we briefly describe three kinds of potential solutions: 1) The link prediction methods may be used to determine whether an edge is noisy or not; 2) Some existing works can provide us some denoising solutions; 3) Deep generative models can learn the distribution of network structure and further regenerate the network.

Link prediction. The link prediction problem [16] aims to predict future possible links in the network. It is feasible to think that link prediction could be used as a method for network denoising, i.e., predicting edge-existence probabilities based on pairwise relationships. The mainstream methods include common neighbors, Jaccard's coefficient, Katz, and latent feature models [17], [18]. However, few of these methods consider non-local network properties. Berton et al. [4] proposed a graph construction algorithm that takes structural information into account, but it ignores the long-term dependence among edges (i.e., whether to generate a new edge is conditioned on the edges have already been generated).

Network denoising models. We also note that there are several works which also study network denoising problem. Dong et al. [38] proposed to learn optimal network structure and node embeddings from the noisy original network for the community detection task. Gu et al. [39] aimed to select a subset of informative links from the original network which enhance the quality of community structures. However,

both of these two works only focused on the community detection task, while were hard to generalize to other tasks. Xu et al. [40] considered the mutual influence between noisy links and missing links, but they can not directly obtain a denoised network. Gao et al. [37] proposed to denoise an individual's social networks by removing unimportant links as noises, but they mainly considered the noisy links in the social media domain and did not take the missing links into account. Another work on the denoising network is Network Enhancement [31], which improves the signal-to-noise ratio of the original networks leading to better downstream performance. Whereas, they updated the noisy edges only by modeling three or less path length, and cannot deal with a longer path dependence. Moreover, this method is mainly applied on biology networks in a limited scale.

Deep models. Along with the recent progress in deep learning on networks [14], a number of deep network generation models have been proposed [8], [9], [11], [12], [13]. Li et al. [9] proposed a graph generative framework based on graph nets. However, they can only deal with small graphs (e.g., ≤ 40 nodes) due to the high memory and time cost. You et al. [12] proposed a RNN-based hierarchical generative model. Bojchevski et al. [13] proposed NetGAN to generate graphs via random walks. Wang et al. [33] proposed GraphGAN, where the generator fits the underlying true connectivity. However, these methods cannot directly perform optimization on desired objectives. Besides, they do not consider the feedback provided by downstream task, which is a valuable feedback for network regeneration. Moreover, hardly of these network generation methods have guaranteed that the intermediate representation is a denoised network with regard to the original network, but our model does.

There are some previous attempts to apply reinforcement learning to learn relation classification from noisy data. The closest work to ours is Jun et al. [1], which formulates instance selection as a reinforcement learning problem. However, this method focuses on noisy labeling and constructs a cleaned dataset by removing noisy instances, while ours pays more attention on the correction process of noisy links on graph. The difficulty of our method is how to correct our noisy data and formulate this links reconstruction problem on graph via reinforcement learning.

6 Conclusion

NetRL, a novel method for denoising networks, is formulated in the framework of reinforcement learning, along with a carefully-designed reward function to both preserve important topological properties from the original noisy network while also benefiting the downstream task. Experiments suggest that NetRL is capable of denoising networks effectively, with better downstream task performance as well as network properties. We further extend our work to network generation problem and get a surprising result.

One of our intended directions for future is to apply NetRL to more downstream applications, such as link prediction, vertex clustering, recommendation systems and so on. Another interesting direction would be to extend our method to directed and weighted networks, and further study its the ability to be generalized to dynamic networks or heterogeneous networks. Moreover, it will be interesting to see the denoised network can be transferred to different downstream task in a more general way.

Acknowledgments. This work is supported by the National Key Research and Development Project of China (No. 2018AAA0101900), the Major Scientific Project of Zhejiang Laboratory (Grant No. 2020MC0AE01), the Fundamental Research Funds for the Central Universities (Zhejiang University New Generation Industrial Control System (NGICS) Platform), the Zhejiang University Robotics Institute (Yuyao) Project (Grant No. K12001). Jiarong Xu's work is supported by Tongdun Technology.

References

[1] J. Feng, M. Huang, L. Zhao, Y. Yang, and X. Zhu, "Reinforcement learning for relation classification from noisy data," in *AAAI'18*, 08 2018.

[2] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[3] X. ming Wu, Z. Li, A. M. So, J. Wright, and S. fu Chang, "Learning with partially absorbing random walks," in *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 3077–3085.

[4] L. Berton, J. Valverde-Rebaza, and A. de Andrade Lopes, "Link prediction in graph construction for supervised and semi-supervised learning," in *IJCNN'15*, 2015, pp. 1–8.

[5] B. Samanta, A. De, N. Ganguly, and M. Gomez-Rodriguez, "Designing random graph models using variational autoencoders with applications to chemical design," *CoRR*, vol. abs/1802.05283, 2018.

[6] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD '14*, 2014, pp. 701–710.

[8] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," *CoRR*, vol. abs/1802.03480, 2018.

[9] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *CoRR*, vol. abs/1803.03324, 2018.

[10] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *CoRR*, vol. abs/1607.00653, 2016.

[11] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop on Bayesian Deep Learning*, 2016.

[12] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: A deep generative model for graphs," *CoRR*, vol. abs/1802.08773, 2018.

[13] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *Proceedings of the Thirty-Fifth International Conference on Machine Learning*, 2018, pp. 610–619.

[14] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

[15] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *CoRR*, vol. abs/1709.05584, 2017. [Online]. Available: http://arxiv.org/abs/1709.05584

[16] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *J. Am. Soc. Inf. Sci. Technol.*, vol. 58, no. 7, pp. 1019–1031, May. 2007.

[17] K. Miller, M. I. Jordan, and T. L. Griffiths, "Nonparametric latent feature models for link prediction," in *Advances in Neural Information Processing Systems 22*. Curran Associates, Inc., 2009, pp. 1276–1284.

[18] A. K. Menon and C. Elkan, "Link prediction via matrix factorization," in *ECML PKDD'11*, 2011, pp. 437–452.

[19] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, "Mixed membership stochastic blockmodels," *J. Mach. Learn. Res.*, vol. 9, pp. 1981–2014, Jun. 2008.

[20] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *J. Mach. Learn. Res.*, vol. 11, pp. 985–1042, Mar. 2010.

[21] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR'14*, 2014.
[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, pp. 2672–2680.
[23] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, May. 1992.
[24] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.
[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR'17*, 2017.
[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[27] R. Feng, Y. Yang, W. Hu, F. Wu, and Y. Zhuang, "Representation learning for scale-free networks," in *AAAI'18*, 2018, pp. 282–289.
[28] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Tech. Rep., 2002.
[29] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, May. 1992.
[30] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI'16*, 2016, pp. 2094–2100.
[31] B. Wang, A. Pourshafeie, M. Zitnik, J. Zhu, C. D. Bustamante, S. Batzoglou, and J. Leskovec, "Network enhancement: a general method to denoise weighted biological networks," *Nature Communications*, 2018.
[32] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," ser. IJCAI'15, 2015, pp. 2111–2117.
[33] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," *CoRR*, vol. abs/1711.08267, 2017.
[34] Y. Hu, "Efficient, high-quality force-directed graph drawing," *Mathematica Journal*, pp.37–71, 2005.
[35] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, no. 1, pp. 47–97, Jan. 2002.
[36] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio and X. Bresson, "Benchmarking graph neural networks," *CoRR*, vol. abs/2003.00982, 2020. [Online]. Available: http://arxiv.org/abs/2003.00982
[37] H. Gao, X. Wang, J. Tang and H. Liu, "Network denoising in social media," *ASONAM'13*, 2013, pp. 564–571.
[38] Y. Dong, M. Luo, J. Li, D. Cai and Q. Zheng, "LookCom: Learning Optimal Network for Community Detection," *IEEE Transactions on Knowledge and Data Engineering*, 2020.
[39] Q. Gu, M. Luo, C. Aggarwal and J. Han, "Unsupervised link selection in networks," *Artificial Intelligence and Statistics*, 2013, pp. 298–306.
[40] J. Xu, Y. Yang, C. Wang, Z. Liu, J. Zhang, L. Chen and J. Lu,"Robust Network Enhancement from Flawed Networks ," *IEEE Transactions on Knowledge and Data Engineering*, 2020.
[41] Y. Yang, Y. Xu, C. Wang, Y. Sun, F. Wu, Y. Zhuang and M. Gu,"Understanding Default Behavior in Online Lending," *CIKM'19*.

Jiarong Xu received her BS degree in information science and technology from the Donghua University, China, in 2016. She is currently working toward the Ph.D. degree in college of control science and engineering at the Zhejiang University. Her current research interests include data mining algorithms, social network theories and reinforcement learning.

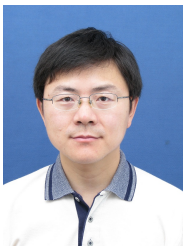
Yang Yang received his Ph.D. degree from Tsinghua University in 2016. He is an associate professor in the College of Computer Science and Technology, Zhejiang University. His main research interests include artificial intelligence in networks and deep learning for large-scale dynamic time-series. He has published over 30 research papers in major international journals and conferences including: TKDE, KDD, WWW, AAAI, and TOIS.

Shiliang Pu received his Ph.D. degree from Zhejiang University. He is the chief expert of Hikvision and the director of Hikvision Research Institute. He has received the Qiu Shi outstanding youth achievement transformation award from the China Association for Science and Technology (CAST). His main research interests include artificial intelligence and large visual data.

Yao Fu received his Ph.D. degree from Zhejiang University. He is a senior engineer in the big data intelligence department, Hikvision Research Institute. His main research interests include knowledge graph and recommender systems.

Jun Feng received her Ph.D. degree from Tsinghua University in 2018. She is an algorithm engineer of State Grid Zhejiang Electric Power Co., LTD. Her research focuses on knowledge graph, reinforcement learning and natural language processing. She has published papers in top conferences like AAAI, WWW and COLING.

Weihao Jiang received his Ph.D. degree from Chinese academy of social science. He is a senior algorithm director of the big data intelligence department, Hikvision Research Institute. His main research interests include data mining and knowledge graph.



Lu Jiangang received the BS and PhD degrees from the Zhejiang University, in 1989 and 1995, respectively. He is currently a professor in the College of Control Science and Engineering, Zhejiang University. His research interests include artificial intelligence and industrial automation.

Chunping Wang received her Ph.D. degree in Machine Learning from Duke University. She started her professional career in Opera Solutions as a Data Scientist, and currently she is a Principal Data Scientist in Fivolution Group. Her current interests include the mining of both structured and unstructured data via machine learning technology to empower the financial industry.