Maximize information extraction from a social graph with Reinforcement Learning

Samir Garibov
Department of Computer Science
Univerity of Freiburg
Freiburg, Germany
garibovs@cs.uni-freiburg.de

Premraj Thakur
Department of Computer Science
Univerity of Freiburg
Freiburg, Germany
premrajscribbles@gmail.com

Thejas Babu Sai Babu
Department of Computer Science
Univerity of Freiburg
Freiburg, Germany
babut@tf.uni-freiburg.de

***Abstract*—Our project focuses on solving extracted information maximization problem from a social network. In this report we first define the information extraction from a social network problem, then show our model of the social network, and finally show how Reinforcement Learning (RL) can be used to solve this problem. The code can be found on our github repository[1]**

***Index Terms*—Reinforcement Learning , DeepRL, Influence Maximization , Information Extraction , Ray, Proximal Policy Optimization**

I. Introduction

In today's fast changing world , to progress fast we need to absorb a lot of information. With the ever-increasing ease of access to internet , the connectivity between individuals and the exchange of information is surpassing all the bounds. However, with the high volume of information it is equally tedious to go through each and every information source. In such a setting , redundancy of information is inevitable as well. As such, to optimally extract all the information it would be judicious to connect to information sources which share original and authentic messages. For example, Twitter happens to be a widely used and powerful platform where information exchange of the order of zillions take place in a short span of time and an average human can fetch information from his twitter feed by engaging in a limited number of scrolls. It would be wasteful in terms of memory and time ,if a person keeps getting the same message from the different accounts he follows and the same if he could not retrieve significant amount of information from his social network. Apart from the world of internet, this problem is worth investigating in the context of real social world as well. Hence, we model this problem of efficiently extracting information by establishing connections as a graph optimization problem.

II. Scope of Reinforcement Learning in Social Network Problems

Reinforcement Learning has been successfully applied to many games like go and chess. The afore-mentioned problem has the similar elements of deciding in the midst of trade-offs and balancing exploration and exploitation. The objective can be seen as a game, in which, the moves of the player are to choose a node and establish connection in such a way that in the long run, it can absorb maximal portion of the information available in the environment. As such it motivated us to phrase this as a reinforcement learning problem.

While going through the body of existing works in this area, we found that RL has been demonstrated to yield good results in influence Maximization Problems, which have the same objective as in our problem but in the reverse direction. The sub-section below throws light on how RL has been applied to Influence maximization and the ideas we drew inspiration from.

A. Social influence maximization problem

Social influence maximization problem intends to find a subset of nodes through which information or influence can be disseminated optimally. In [2],[3],[4] ,Reinforcement Learning is applied to optimize the spread of influence in a social network. broadly speaking , the setting of the IM problem involves selecting a node at each timestep as an action,and collecting a reward as a function of the number of nodes which could be directly influenced through the newly added node. Both [3] and [4] use deep-Q algorithm to learn optimal policies although [4] combines GCN along with deep-Q learning. both [3] and [4] have been inspired by [2], which used Influence Maximization to choose individuals among a community of community of homeless youth to propagate information about HIV. In their setup, each chosen individual had a certain probability to accept an invitation to an intervention and a certain probability to act as a seed node to propagate information to individuals connected to him/her. The number of individuals to whom the information could reach was chosen as the reward.

B. Maximal Information Extraction Problem

In our problem setup , we have one information seeker who is in an environment of a number of information publishers. At each move, it can either choose to not connect to any of the unconnected nodes or it can choose two nodes , one to connect to and one to disconnect. Once it makes its move, it pulls information in the form of messages from all the nodes it is connected to. The reward function is chosen such that

[1]https://github.com/karibbov/infoMax

it encourages capturing a higher proportion of the number of unique messages being circulated in the environment. When framed in this way, the problem of information extraction becomes conducive to the technique in the field of Reinforcement Learning.

III. Methodology

A. Social network model

In this section we explain the social network model we used to simulate social networks.

To evaluate and train our agent to solve the Information Extraction problem we first needed to simulate the real-life interactions on a social network, which is a quite complicated and chaotic system. Therefore, we made some assumptions and simplifications while trying to preserve the core dynamic of social networks.

We represent the social network as a directed graph. where the nodes represent the independent users and the directed edges represent the flow of information between two users. In other words, edges represent the followee-follower relationship, directed from the former to the latter. Each directed edge also holds a single message object that represents a post shared by a followee to a follower. At each time step the nodes can either share an original unique message they created or re-share a message they received from their followees. Additionally, if the *dynamic* flag is set then a node can also decide to follow or unfollow another node by creating or removing a directed edge from another to itself. To simulate the interactions realistically and keep the implementation simple, we let each node make decisions independently and stochastically. Therefore, besides a unique identifier each node has 4 different probabilities characterising their decisions: *Originality*, *Probability of Following*, *Probability of Unfollowing*, *Probability of Randomly following*.

Originality defines the probability of the node sharing a unique message as opposed to re-sharing a message it received from a parent node. *Probability of Following* defines the probability of the node following another node that's higher up in the graph than itself (i.e. parents, parents of parents etc). *Probability of Unfollowing* defines the probability of the node unfollowing a parent node. *Probability of Randomly following* defines the probability of randomly adding an edge directed from any node to itself. In case there are leaf nodes in the graph the node will follow a leaf node first, and if there are none then it will default back to following any node. Although, this parameter usually is set to a very low value (=0.05 on our experiments) it is designed to increase the connectivity of the graph and make it easier for the information to propagate.

As mentioned earlier each directed edge holds a message object that is propagated from a parent node to its successors. Each new message object holds a unique signature, information about the node that created this message and a value deciding how likely it is to be shared by a successor node. So if a successor node decides to share one of the messages it received, it will share the one with the highest likelihood of sharing value.

In Figure 1 and 2 we demonstrate a simple case of our model with 6 nodes. Note that although with lower originality probabilities it would be less likely for a node to share an original message it would still be possible, but a node with no parent node would only share original information regarless of its originality parameter.

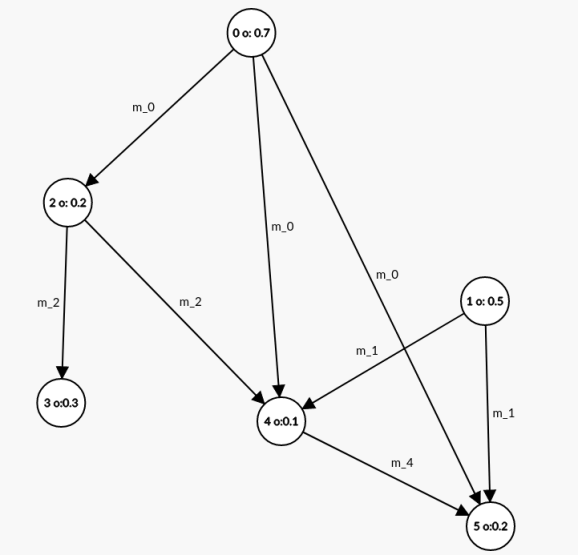

Fig. 1. A simplified snapshot of the social network model at step 0, o : represents the originality parameters of the nodes, and m_i represents a message originating from i^{th} node

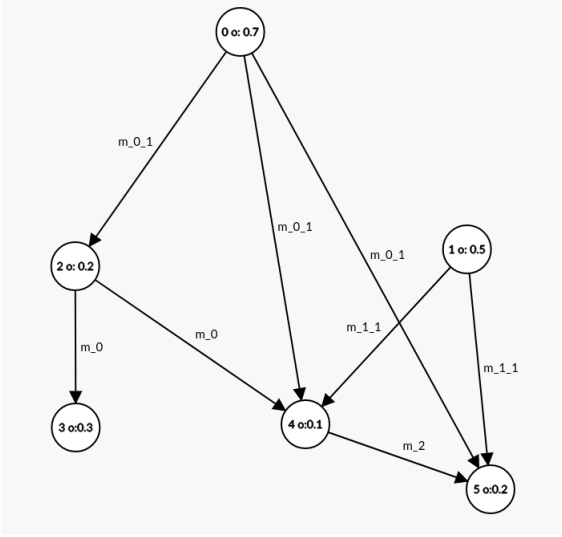

Fig. 2. A snapshot of the same network at step 1, here the second index on the message shows the step the message was created on. On this step node 2 decided to re-share the information it received from node 0, node 4 decided to re-share the message received form node 2 and node 0 and 1 created new messages since they don't receive anything

B. Reinforcement Learning framework

Discuss our action/observation space, reward function and evaluation function here. To train a Reinforcement Learning algorithm on our model, we implemented a custom training environment in which we define our action space, observation space, and reward function. In this subsection we will define our implementation of each and explain our reasoning for the design decisions.

Since the implementation of a graph based observation space was a rather complicated task, we decided to not include the graph representation of our social network in the observation space our agent. Therefore we defined 3 different lists to consist our observation space, these lists hold the information about the: Connected and unconnected nodes, Number of unique messages coming from a node mapped to the parent node of the agent, Number of unique messages coming from a node mapped to the original node creating that message. First and second lists are holding information about the edge between our agent and the social network i.e. information about the nodes our agent is already connected to. Whereas, the third list can hold the information about any node in the paths leading to the agent (i.e. any parent or parent of parent etc.). Moreover, second and third list holds historic information as well, as they record the number of all the unique messages through a run.

For the action space initially we had 2 different candidates, the most intuitive option would be to have a different discrete action to connect to or disconnect from each node in the given network, in this case we have n different actions for choosing a node to connect to, further n more to choose a node to disconnect from also 1 more action for each for *noOp*. Since the choice of the node to connect to and disconnect from are independent, the total action space would have the dimensionality of $(n+1)^2$. Which depending on the reinforcement learning algorithm can me too much to optimize with. To decrease the dimensions of the action space we also considered using 2 actions, one for choosing the n^{th} parent of the agent node and the other one for choosing which of the parents from this layer to connect to. Although, this would decrease the size of our action space, it would also make the implementation more complex and require implicitly giving the graph information to the agent, which we don't want to do as discussed earlier. Therefore, we only tried the intuitive approach which generated acceptable results.

The reward function is defined as the ratio of the difference between the unique and duplicate messages the agent receives in one step and the total number of messages sent by all nodes across the network in one step. Let's say at step t all nodes in the network sends N_t message and the agent receives m_t messages and k_t of them are never seen by our agent before. So the reward function $R(s)$ for the step t will be:

$$R(t) = \frac{2k_t - m_t}{N_t} \tag{1}$$

For the non dynamic social network case where the connection between the nodes in the network does not change, N_t remains stable during the whole run and is equal to the number of the nodes which has a successor node connected to them. Since each node can share only one message at one step, the number of received messages m_t is equal to the number of the connected nodes.

From here on, we will loosely use the the term approximation as a replacement for getting an expected value of, to declutter both the notation and the text. To analyse lets say the originality probability of the i^{th} node $node_i$ is o_i, which can be approximated by the agent as $o_i \approxeq \frac{k_i}{m_i}$ where k_i and m_i are the original and total messages shared by the node over the whole run. Lets further define the total reward R_total as $R_{total} = \sum_{t=1}^{T} R(t)$ where T is the final number of steps taken during the run. With these we can approximate the contribution of the i^{th} node to the total reward across all of the steps.

$$R_{total}(i) \approxeq \frac{(2o_i - 1)m_i}{N_t} s_i \tag{2}$$

Here, s_i is the number of steps $node_i$ was connected to the agent. We can further dissect this by dividing the contribution of $node_i$ by s_i and since each node can only send a single message in each step m_i is equal to 1 for all $node_i$. We can now define our reward function as a sum of contributions of each node:

$$R(t) \approxeq \sum_{i=1}^{m_t} \frac{2o_i - 1}{N_t} \tag{3}$$

Thus we can see that, the optimal policy would be to connect to all the nodes with the originality probability $o_i \geq 0.5$.

C. Reinforcement Learning algorithm

We have experimented with PPO(Proximal Policy Optimization) as the RL algorithm of our choice for this use-case. A policy is a mapping from the action space to state space. RL algorithms aim to find out the best policy,which governs the actions to be taken at a certain state, that would lead to the maximization of the cumulative reward. In Deep RL, Policy Gradient method ensures that the parameters of the neural network,being used to approximate the policy, shifts towards the optimal policy which yield higher cumulative rewards.

$$\nabla_\theta E_x[f(x)] =$$
$$\nabla_\theta \sum_x p(x) f(X) = E_x[f(x) \nabla_\theta \log p(x)]$$
$$f(x) : Reward\ Function\ or\ Score\ Function$$
$$p(x)\ :\ function\ approximated\ by\ Policy\ network,$$
$$which\ gives\ the\ probability\ of\ taking\ a\ certain\ action,$$
$$X,\ given\ certain\ state,\ S$$

Mathematically, to facilitate SGD , the gradient of an expectation is converted into an expectation of a gradient. Proximal Policy Optimization(PPO) algorithm falls under the class of Policy Gradient algorithms in which the deviation of the

Policies in between subsequent iterations is intended to be controlled. Trust Region methods and methods with modified objective functions such as clipped surrogate objective function and adaptive KL penalty coefficient fall under this class of algorithms. Proximal Policy Optimization Algorithm as mentioned in [6] ,where it was propounded , have the data efficiency and reliable performance of Trust Region methods in spite of using first order methods. PPO tries to maximize the following objective function

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t]$$

$$with$$

$$r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$$

The second term clip in the above objective function modifies the objective function by clipping the probability ration which removes the incentive for moving r_t outside $(1-\epsilon, 1+\epsilon)$

IV. Results and Discussion

One of the main questions before running the experiments was how to initialize the social network model. To keep the social network realistic but still simple enough for the algorithm to be able learn, we decided to use 50 nodes, and run agent on each network for 5000 steps. We also ran the algorithm on both dynamic and static networks for each node parameter we randomly sampled the *Originality*, *Probability of Following* and *Probability of Unfollowing* from a gamma distribution with the scale parameter of 2 and the shape parameter of 3, before initializing the agent we also ran the simulation for 300 steps in the dynamic setting to get a more natural social network graph.

To evaluate the final result of the runs we used the percentage of the unique messages(PUM) in the total messages received across the run through all steps. So the evaluation metric:

$$PUM = \frac{k_{total}}{m_total} \tag{4}$$

Where the k_{total} and m_{total} represent the total unique messages received and the total messages received respectively.

With the above settings we ran the PPO algorithm for 30 training epochs and got the maximum PUM of 0.80, where in the case of agent fully connecting to the network the PUM would be 0.44. To further demonstrate the ability of the algorithm for approximating the originality values of the nodes from the observations we plot the originality values of the selected nodes in Figure 3

We furthermore conducted the same experiments under the dynamic setting of the social network model. Although, the performance was better than selecting randomly PUM was significantly less than the static case with the PUM of 0.68.

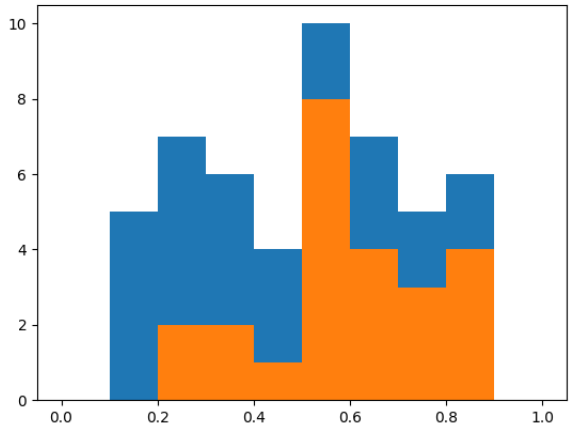

Fig. 3. The blue bars show the distribution of the originality probabilities across the whole network, the orange bars show the number of the originality probabilities of the nodes chosen by the final algorithm. We can see that the algorithm favors the nodes with originality values higher than 0.5

V. Conclusion

This initiative to apply reinforcement learning for information extraction , which is in opposite direction of influence maximization shows good promise on the effectiveness of RL as a solution here. We envisage to extend this to more dynamic setting and multi-factored modelling of social interaction.A generalization to a Multi-agent Reinforcement Learning Scenarion will be worthwhile to try to produce even more realistic solutions.

References

[1] Yilei Hu, Brian Skyrms, Pierre Tarrès ,"Reinforcement Learning in Social Networks" , arXiv:1103.5818 [math.PR]

[2] Amulya Yadav, Hau Chan, Albert Jiang, Haifeng Xu, Eric Rice, Milind Tambe, "Amulya Yadav, Hau Chan, Albert Jiang, Haifeng Xu, Eric Rice, Milind Tambe " , arXiv:1602.00165 [cs.AI]

[3] Haipeng Chen, Wei Qiu, Han-Ching Ou, Bo An, Milind Tambe , 'Contingency-Aware Influence Maximization: A Reinforcement Learning Approach", arXiv:2106.07039 [cs.SI]

[4] Harshavardhan Kamarthi, Priyesh Vijayan, Bryan Wilder, Balaraman Ravindran, Milind Tambe , "Influence maximization in unknown social networks: Learning Policies for Effective Graph Sampling", arXiv:1602.00165 [cs.AI]

[5] Harshavardhan Kamarthi, Priyesh Vijayan, Bryan Wilder, Balaraman Ravindran, Milind Tambe , "Influence maximization in unknown social networks: Learning Policies for Effective Graph Sampling", arXiv:1602.00165 [cs.AI]

[6] Dhariwal, Alec Radford, Oleg Klimov , "Proximal Policy Optimization Algorithms", arXiv:1602.00165 [cs.AI]

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. ArXiv Preprint ArXiv:1606.01540.

[8] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., ... and Stoica, I. (2018, July). RLlib: Abstractions for distributed reinforcement learning. In International Conference on Machine Learning (pp. 3053-3062). PMLR.