Reinforcement Learning Techniques for Graph Classification

Isaiah King
April 2020

I. Introduction

Graph learning and classification has long been a notoriously difficult problem. The problem domain is not in Euclidian space, as nodes on graphs may have a functionally infinite degree. Further, there are the problems of graphs' irregular structure, heterogeneity of graphs in practice, and the large scale, and combinatorial growth in complexity of graph algorithms. As such, most graph problems are NP-hard and extremely computationally expensive to solve in any sound way. Lately, there have been great strides using machine learning techniques, and reinforcement learning to approximate solutions to graph problems.

Finding an effective method of deep learning on graphs has been described as, "the crown jewel of artificial intelligence and machine learning" [1]. In order to find this jewel, we present several RL-based approaches for graph or node classification. We categorize these approaches as either deep reinforcement learning–meaning the mechanics of how the technique works is a black-box–or explainable reinforcement learning. Both approaches have their own benefits, and both have several working implementations, however, in this paper, we will focus more on the explainable approaches than deep reinforcement learning. These approaches have more in common with traditional reinforcement learning, while the latter has more in common with traditional deep learning.

II. Deep RL

We define **deep reinforcement learning** (D-RL) as any reinforcement learning approach which uses a neural network to encode states and feeds these states into still another network which acts as the policy. In theory, these methods can still be defined in traditional reinforcement learning terms. However, often it is simply a matter of training a neural network using a reward function in lieu of a loss function. The definition used by [2] defines D-RL networks as traditional reinforcement learning which incporperates a Q function, $\mathrm{Q}(s, a)$ which maps every tuple $< s, a >$ to a vector $V^d \in \mathbb{R}^d$. This vector is then used by some policy function to determine the next action and the reward.

The reason for mapping state-action pairs to vectors is that often, the state-action space may be unreasonably large, so tabular methods are infeasible. This approach appears to be the most popular, as all but one method mentioned in the reinforcement learning section of survey [1] follows this method. Of these D-RL papers, only [3] is a method of graph classification, however.

Graph Classification using Structural Attention [3] describes a D-RL method that classifies heterogeneous graphs based on minimal length random walks through the graph. Formally, the problem is defined as follows: a graph $G = \{V, E\}$ where V is a vertex which maps to a certain type, $V \to t \in T$ and G has a certain label, $l \in L$. The agent attempts to use a guided random walk to determine l for a given graph.

The agent achieves this by using an attention network as the policy, as well as the Q function. The algorithm builds a vector, S, through a linear combination between a history vector h and a vector representation of the current node it is exploring. This combination can be thought of as the state vector. The set of actions is the set of the current node's neighbors it shall explore next, and the reward is determined by both the length of the current walk, and if the neural network interpreting a combination of the current node and a history vector can correctly classify the graph. A linear combination of S and h, $\hat{h}$, is fed into a neural network. If the network classifies the graph correctly it receives a reward. Otherwise $\hat{h}$ is used as the next history vector, and a new node is explored. The policy for which node to explore next is an attention network which uses $\hat{h}$ and S to determine what kind of nodes would be

most beneficial to explore next.

On the surface, this does seem like a well-defined reinforcement learning model. However, the execution is just a standard attention network described with the trappings of a reinforcement learning problem. This is the main issue we take with D-RL methods: they lean so heavily on recursive neural networks, convolutional neural networks, or graph neural networks, that they often are only tangentially related to reinforcement learning. Certainly, the efficacy of these methods is not in question; they often produce very good results very quickly. But for true reinforcement learning techniques to tackle the challenges of graph structures, we turn to explainable reinforcement learning.

III. Explainable RL

Explainable reinforcement learning (E-RL) techniques for graph classification are disparate, and often a means to solve a tabular reinforcement learning problem by using the abstraction of a graph. However, this does not mean they can't be applied to graph problems generally. One technique used for abstracted problems, but that can be applied generally is the idea of a centrality measure. This defines the relative importance of a node in a graph [2]. It's not unlike the concept of a state value function, $V_\pi(s)$ which approximates the average reward that comes from transitioning to a given state. In some graph based reinforcement learning approaches, the centrality function of a node is used as the reward function itself; it can be a very powerful tool.

A. Centrality Measures

The simplest measure of node centrality in a graph is *degree centrality* [2]. This is simply the number of neighbors a given node has. Formally, the degree centrality of a node n is defined as

$$C_d(n) = |(n, v) \in E|$$

This measure is simplistic, and as a result it is rarely used on its own. More often, centrality measures use it as a starting point to build from.

Another naive centrality measure is *closeness centrality* [2]. This is defined as the inverse of the mean distance between a node n and all other nodes in the graph. Formally it is defined as

$$C_c(n) = \frac{n-1}{\sum_{m \in V \wedge m \neq n} \delta(m, n)}$$

where $\delta(m, n)$ is the distance between nodes n and m. This method is effective, and sometimes used on its own, but it is difficult to scale. The distance between two nodes is expensive to calculate, and the necessity to calculate it for every pair of nodes makes this approach undesirable on larger graphs.

A more advanced method of measuring node centrality are measures of *novelty* and *relative novelty* [4]. This approach simulates several random walks through the graph. Each time a node is visited, it is marked. The node's novelty is calculated by

$$C_n(n) = \frac{1}{v}$$

where v is the number of visits to node n. This method prioritizes exploring nodes that have not yet been visited. This works well for the exploratory phase of a policy, but for a greedy phase, more information is needed. This is where the relative novelty score comes into play: this measures the ratio of nodes that precede and follow a node in a given path. Formally, it can be described as

$$C_{rn}(n) = \frac{\sum_{v \in B} C_n(v)}{\sum_{u \in A} C_n(u)}$$

where B is the set of nodes preceding n on the walk, and A is the set of nodes walked after traversing A. The size of these two sets–or the "novelty lag"–is a hyperparameter to be tuned, but Simsek & Barto found 7 is often effective.

Connection Bridge Centrality [5] is an even more advanced centrality measure which builds upon the techniques discussed by [2]. This method uses a combination of two main centrality measures, which both build upon simpler ones. First, it determines the *node connection graph stability*. This measures the number of shortest paths that must flow through a given node, or what can be thought of as how much of a bottleneck a given node is in the flow of a network. Formally this measure is defined as

$$NCGS(n) = \sum_{u \neq v \in V} \frac{d_{u,v}(n)}{\sigma_{u,v}(n)}$$

where $d_{u,v}(n)$ is the length of the shortest path connecting u and v that passes through n, and $\sigma_{u,v}(n)$ is the number of such paths.

Next, Moradi et al. propose a novel technique to add more information to the NCGS score: the bridge centrality score. Because NCGS is mostly a measure of a node's global state, [5] argues that there should

be some consideration for a node's local state as well in assigning it a score. For this reason, they define bridge centrality as a function of a node's degree as well as its one-hop neighbor's degrees. Formally, it is defined as

$$BR(n) = e^{\alpha|d(n) - \frac{1}{d(n)} \sum_{v \in \mathcal{N}(n)} d(v)|}$$

where $d(n)$ is the degree of node n and $\mathcal{N}(n)$ is the set of nodes which neighbor node n. Finally, connection bridge centrality can be found by taking the product of NCGS and BR.

This technique has been shown to be very effective at identifying nodes which have high average reward. This is because it identifies nodes which compared to their neighbors more often are part of a shorter path to any other given node. Though the technique is effective, it is highly expensive to calculate. Finding the shortest path between two nodes is an NP-hard task, and it must be done for every node pair in the graph to determine centrality effectively. However, in smaller sample spaces, this is easily the best centrality measure.

B. E-RL Graph Classification

Now that the notion of a centrality score is well defined, we introduce a method which uses a combination of D-RL techniques and E-RL. The *DeepPath* algorithm [6] aims to use E-RL techniques to discover unique paths through a knowledge base. While a knowledge base is a highly specialized data structure, it can nevertheless be thought of as a highly heterogeneous graph. This technique defines several explainable policies which take heavy influence from centrality measures which it then uses to guide walks which are vectorized in a D-RL manner.

The algorithm defines states as a vector encoding of the current node being explored. Unlike [3], there is no notion of the walk's history in this encoding; it is stateless. It then uses this encoding to determine which neighbor it should walk to. This is accomplished by first training the network to select actions which lead to exploring nodes which are connected to the current one in general. To expedite this process, the authors adopted an imitation learning technique and had the agent learn from successful BFS paths between two nodes. This training only allows the agent to find neighbors based on its state encoding. In effect, it learns the possible actions from a given state. At this stage the agent is not making very informed decisions. This is where the E-RL techniques come into play.

Once the agent can successfully identify possible next states from its current state encoding, a reward function is introduced, or rather, several are. The agent is rewarded for *global accuracy*, *path efficiency*, and *path diversity*. Global accuracy is simple: a reward of 1 if the agent reaches a target node, 0 otherwise. Path efficiency is also quite simple, additional reward is added based on the inverse length of the path to reward shorter paths. Finally, path diversity is a function of the sum of all encodings of nodes in the path found. It is calculated as follows:

$$r_{\text{diversity}}(p) = \frac{1}{|F|} \sum_{i}^{|F|} \cos(p, p_i)$$

where F is the set of found paths, and $\cos(p, p_i)$ is the cosine similarity between the sum of states in the paths p and $p_i \in F$. With these reward functions, the policy is then defined by two fully connected layers which map states to a probability distribution of possible actions. The loss function is then defined as a gradient of the expected reward which updates after every completed path, making this a Markov Decision Process. We observe that the use of these very explainable reward functions is not at all unlike the notion of a centrality measure. In fact, the path diversity function is a continuously updating centrality measure of all nodes considered before an action. Unlike other centrality-based measures discussed, however, this method uses centrality as a negative reward, instead opting to discover new nodes.

IV. Future Work

Though [6] only aims to find novel and "useful" paths, we believe this technique could be applied to other graph classification algorithms. For example, the *node2vec* algorithm, which classifies graphs based on random walks through the graph [7]. We believe if these walks were guided by a policy in a similar way to *DeepPath* the resulting node embeddings would be more useful as the walks would carry more information. For this reason, we intend to implement a centrality-score guided walk process to inform the *node2vec* algorithm. *node2vec* is based upon a skip gram designed to classify word relationships. It works by feeding

sequences of random walks into this skipgram as if explored nodes were words in a sentence. It uses this information to construct node embeddings in the same way *word2vec* does with text data [8]. We believe prioritizing exploration of novel nodes will be an effective method to generate better node embeddings. By combining the idea of a centrality score as an expected reward function, and the path scoring methods from *DeepPath*, a Markov Decision Process that guides *node2vec* will generate more informative node embeddings.

With more informative node embeddings, graph classification problems become simpler. Obviously node classification with vector representations of each node is a simple task requiring no more than standard machine learning techniques to accomplish, but graph classification overall can be made far simpler as well. With a network of vectors, auto encoding methods of graph classification [1] or even simple methods such as sums of paths as [6] does become far more informative. This technique could be used for anything from social media mining to developing new drugs.

Graph classification represents an elegant solution to many complex problems that have been until recently too computationally expensive to answer quickly. By using reinforcement learning methods, we hope to find efficient approximations that answer these problems. There is still much work to be done, but we believe the crown jewel of machine learning is close at hand.

References

[1] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, p. 14–14, 2020.

[2] M. R. F. MendonÇa, A. Ziviani, and A. M. S. Barreto, "Graph-based skill acquisition for reinforcement learning," *ACM Comput. Surv.*, vol. 52, no. 1, Feb. 2019. [Online]. Available: https://doi.org/10.1145/3291045

[3] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, 2018.

[4] O. Şimşek and A. G. Barto, "Using relative novelty to identify useful temporal abstractions in reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 95. [Online]. Available: https://doi.org/10.1145/1015330.1015353

[5] Moradi, Shiri, and Entezari, "Automatic skill acquisition in reinforcement learning agents using connection bridge centrality," in *Communication and Networking International Conference, FGCN 2010 Held as Part of the Future Generation Information Technology Conference Part 2*, ser. FGCN 2010. Springer, 2010, p. 51–61.

[6] W. Xiong, T. Hoang, and W. Y. Wang, "Deeppath: A reinforcement learning method for knowledge graph reasoning," *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.

[7] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 855–864. [Online]. Available: https://doi.org/10.1145/2939672.2939754

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv e-prints*, p. arXiv:1301.3781, Jan. 2013.