We are IntechOpen, the world’s leading publisher of Open Access books
Built by scientists, for scientists

6,600 Open access books available
177,000 International authors and editors
195M Downloads

154 Countries delivered to
TOP 1% Our authors are among the most cited scientists
12.2% Contributors from top 500 universities

WEB OF SCIENCE™
Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com
Chapter

Graph Neural Networks and Reinforcement Learning: A Survey

Fatemeh Fathinezhad, Peyman Adibi, Bijan Shoushtarian and Jocelyn Chanussot

Abstract

Graph neural network (GNN) is an emerging field of research that tries to generalize deep learning architectures to work with non-Euclidean data. Nowadays, combining deep reinforcement learning (DRL) with GNN for graph-structured problems, especially in multi-agent environments, is a powerful technique in modern deep learning. From the computational point of view, multi-agent environments are inherently complex, because future rewards depend on the joint actions of multiple agents. This chapter tries to examine different types of applying GNN and DRL techniques in the most common representations of multi-agent problems and their challenges. In general, the fusion of GNN and DRL can be addressed from two different points of view. First, GNN is used to influence the DRL performance and improve its formulation. Here, GNN is applied in relational DRL structures such as multi-agent and multi-task DRL. Second, DRL is used to improve the application of GNN. From this viewpoint, DRL can be used for a variety of purposes including neural architecture search and improving the explanatory power of GNN predictions.

Keywords: graph neural network, deep reinforcement learning, multi-agent, multi-task, neural architecture search

1. Introduction

Building an intelligent system that can extract high-level representation from data is necessary for many issues related to artificial intelligence. Theoretical and biological arguments show that to build such systems, deep architecture models are needed that include many layers of non-linear processing units. Before the emergence of deep learning [1], traditional machine learning approaches depended on the representations given by feature selection or extraction that get from the data.

These methods required an expert in the domain of the subject to extract the features manually. However, this hand-crafted feature extraction is a time-consuming and sub-optimal process. The emergence of deep learning could quickly replace these traditional methods because it could automatically extract the features according to each problem. In recent years, deep learning has become the main motivation for innovative solutions to artificial intelligence problems. This issue has been made
possible by increasing the amount of available data, increasing computing resources, and improving techniques in training deep networks.

Deep neural networks (DNNs) have reached remarkable achievements in the last decade [2]. However, basic types of neural networks can only be implemented using regular or Euclidean data. Whereas, many data in the real world have a graph structure that is a non-Euclidean data structure. This irregularity of the data structure has led to recent advances in graph neural networks (GNNs).

GNNs [3, 4] allow the creation of a machine learning model which is taught simultaneously to learn the representation of data with a graph structure. GNNs are undoubtedly the most interesting issue in graph-based deep learning. GNNs can be applied to graph-structured data for various tasks, from clustering to classification or regression. They can also learn representation at the level of nodes, edges, and graphs. Deep learning with graph data structure is recognized as graph representation learning, geometric deep learning, or graph embedding which seeks to learn the representation of structured information about graphs.

The purpose of graph representation is to build sets of features that show the structure of the graph and the data in it. The main key of this method is to learn a mapping that embeds nodes or graphs as points in a low-dimensional vector space so that this mapping is optimized and the geometric relations in this learned space reflect the structure of the original graph. After optimizing the built-in space, this learned embedding space can be used as input features of the graph.

Once the size of the dataset in the input network is different from the training history GNNs play a highly efficient role in knowledge transfer between data-oriented structures. GNNs are inherently designed to generalize over graphs of different structures and sizes. This ability allows the GNN-based DRL agent to learn and generalize over arbitrary network of environment topologies. Many DRL methods apply standard neural networks such as recurrent neural networks (RNNs) [5], and other neural network structures. This issue causes poor generalization and prevents the deployment of DRL in networks, because it is hard to adjust to the dynamic changes of network topology. In recent years, the integration of GNNs and DRL specially in multi-agent systems has attracted much attention in graph-structured environments. Nowadays, many systems can be viewed as multi-agent systems from a new perspective. The cooperation of a group of agents (teamwork) in the frame of a graph is one of the most important issues that is always raised in multi-agent systems [6], due to increasing the ability to reach the final goal of the system and improving the overall strategy.

This issue becomes more important when the environment is complex and dynamic. In such an environment, a purposeful agent is affected by the actions of other agents in addition to the changes in the environment. Therefore, the environment has more dynamic states than before, and the agent must have the ability to model the action process and the power to learn and interact with other agents. Using classical methods in describing agents and establishing communication between them in a multi-agent environment, due to the use of many equations, weakens the power of expanding the network to large systems. By defining an intelligent system and using smart modern methods in solving such problems, methods such as deep reinforcement learning algorithms have been proven to be useful in such environments.

Automated control problems and development in decision-making are the results of recent advances in DRL [7]. However, existing DRL-based solutions still fail to generalize when applied to network-related scenarios. So, when faced with a network state that is not seen during training, the ability of the DRL agent is impaired.
Recently, GNNs have been offered to model and operate on graphs to reach combinatorial generalization and relational reasoning. Indeed, GNNs simplify the learning of relations between entities in a graph and the rules for composing them. A combination of DRL and GNN can work and optimize problems while generalizing to unseen topologies. Specifically, the GNN used by the DRL agent is inspired by message-passing neural networks [8].

Robotics, pattern recognition, recommendation systems, and games are some of the subjects in which DRL has presented acceptable performance. On the other hand, GNNs exhibit excellent efficiency in supervised learning for graph-structured data [9]. DRLs utilize the ability of DNNs to solve sequential decision problems with RL, and on the other hand, GNNs are new architectures that are suitable for organizing graph-structured data in this field.

In this survey, an overview of the concepts of GNNs is prepared, and then their relationship with reinforcement learning (RL) is explained. The rest of this chapter is structured as follows. A short review of graph neural networks is given in Section 2. The technical backgrounds of deep reinforcement learning concepts and multi-agent reinforcement learning are presented in Section 3. The relation between RL and GNN is presented in Section 4. Finally, the conclusion is provided in the last section.

2. Graph neural networks

Nowadays, many learning problems need to use graph representation to present the complex relationship between data [10, 11]. Recently, more attention over studies on graph models has been received due to the great expressive power in social science (social networks) [12–14] and biology science (predicting protein interface and bioinformatics analysis, knowledge graphs, modeling physics systems, and classifying diseases) [15–17].

Pairwise message passing is one of the main elements in the structure of GNNs, such that each node in the graph frequently updates its representations by replacing information with its neighbors until a stable balance is attended. The graph neural network usually contains two parts: the message passing part for extraction of local infrastructure features used around the nodes and the readout phase which is an aggregation part to summarize the particular features of the node in a vector of features of the graph surface.

Representing data as a graph has several advantages, such as a simplified representation of complex problems, systematic modeling of relationships, etc. On the other hand, working with data with a graph structure using common DNN-based methods has its own challenges. The variable size of the unordered nodes, the uneven structure of the graph, and the dynamic neighborhood composition make it difficult to implement basic mathematical methods such as convolution on the graph. Graph neural networks (GNN) as its general structure is shown in Figure 1, overcome this defect with the help of new DNN methods in the graph structures of datasets. GNN architectures can model structural information and node features. In the following several well-known models of graph neural networks are introduced:

2.1 Graph convolutional network

For the first time in [18], spectral networks and local deep networks were connected on a graph convolutional network (GCN), as a method for semi-supervised
learning on graph-structured data. The definition of these networks is based on the notion of convolutional neural networks, which are applied to the graphs. GCNs [19] are learned according to the structure of the features of the neighboring nodes. In general, the main difference between CNN and GCN is related to their data structure. CNNs are defined in Euclidean space while GCNs work on graph structure (non-Euclidean structure data) where the number of node connections is different and also the nodes are unordered.

Spatial graph convolutional networks and spectral graph convolutional networks are the two main branches of GCNs. The key idea in spectral GCN was defined by signal/wave propagation. In spectral GCN, information is propagated along the nodes as signal propagation. Eigen-decomposition of graph Laplacian matrix in spectral GCNs is used for information propagation and also is used for node classification by understanding the graph structure. Non-generalization and inefficiency of computations in spectral graph convolutional networks are two main challenges in spectral graph convolutional networks. GCN overcomes these problems by Chebyshev polynomials to approximate the spectral convolution and the ChebNet network is defined [20].

2.2 Graph attention network

Graph attention network (GAT) architecture [21] is a type of GCN architecture in which the aggregation process learns the weights between the neighboring nodes of each node with the help of the attention mechanism. In these networks, greater weights are applied to more important nodes and it stores the weight of the nodes. The advantage of graph attention networks is that these networks can adaptively learn the importance of each neighbor. However, since the attention weights between each pair

Figure 1. Graph neural networks (GNN) framework.
of neighbors must be calculated, the calculation cost as well as the amount of memory occupied increases rapidly.

2.3 GraphSAGE

In graph theory, there is a concept called node embedding, which means mapping nodes to an embedded space with dimensions less than the actual dimension of the data defined on the nodes of the graph, in which similar nodes are embedded close to each other, in the resulting latent space.

GraphSAGE [22] is a deductive learning technique that exploits node features to learn an embedding function for dynamic graphs. This inductive learning approach is scalable across graphs of different sizes as well as subgraphs within a given graph. A new node can be embedded without retraining by the GraphSAGE approach. It uses aggregator functions to induce new node embeddings based on node features and neighborhoods.

In [23] a method for data-driven neighborhood subsampling is defined by a non-linear regressor based on the real-valued importance of each node and its neighborhood. This subsampling helps to embed nodes in the graph using a small set of neighboring nodes with high importance. The regressor is learned using value-based reinforcement learning. Here, the negative classification loss output of GraphSAGE is used to extract this importance.

GraphSAGE-D3QN [24] presents a graph DRL method for emergency control of undervoltage load shedding model. Feature extraction of states in this model is designed by GraphSAGE-based method with topology variation in the training step and then online emergency control is achieved.

2.4 Applications of GNNs

Link prediction [25], node classification [26], clustering [27] and, etc., are considered as graph analysis objectives. In the following, several common GNNs goals are described:

**Node classification**: training models to classify nodes by determining the label of samples that are shown as nodes. Usually, these problems are used in a semi-supervised way, with only a part of the graph being labeled.

**Graph classification**: Graph classification is a task with real applications in social network analysis, categorizing documents in natural language processing, and classifying proteins in bioinformatics fields. Graph classification obtains a graph feature that aids discriminate between graphs of different classes.

**Graph Visualization**: Visual representation of data structures and anomalies with the help of geometric graph theory and information visualization that helps the user understand graphs.

**Link prediction**: Predicting the relationship between two nodes and considering that nodes in a network are likely to have links. An application of this approach is to detect social interactions or suggest potential friends to users on social networks. It has also been used in predicting criminal associations, and in recommender system problems.

**Graph clustering**: clustering on graphs is performed in two ways. Either clustering is based on nodes that should be converted into different and connected groups based
Deep Learning and Reinforcement Learning

on the edge distances and their weights or considering the graph as objects that should be clustered, and clusters these objects based on similarity.

3. Deep reinforcement learning

Using DNNs to solve sequential decision issues in the framework of RL led to the emergence of deep reinforcement learning (DRL) in high-dimensional problems (see Figure 2). Nowadays, different applications of artificial intelligence have been enhanced with the help of DRL which includes natural language processing [28], transportation [29], finance [30], healthcare [31], robotics [32], recommendation systems [33], and gaming [34]. DRL can be defined as a system that maximizes the long-term reward in a reinforcement learning problem using representations that are themselves learned by the deep network. The outstanding success of DRL can be considered due to the ability of this method to deal with complex problems and provide efficient, scalable, and flexible computational methods. Also, DRLs have a high ability to understand the dynamics of the environment and produce optimal actions according to their interactions with the environment. When dealing with various high-dimensional problems or continuous states, reinforcement learning suffers from the problem of inefficient feature representation. Therefore, learning time is slow and techniques should be designed to speed up the learning process. The most significant feature of deep learning is that DNNs can discover compact representations of high-dimensional data automatically.

Combining DNNs with RL has become more attractive in recent years and it has gently shifted the focus from single-agent environments to multi-agent ones. Working

Figure 2.
Total structure of the combination of GNNs and DRL.
with multiple agents is inherently more complex because future rewards depend on the joint actions of several agents and the computational complexity of the function increases. Single-agent environments such as Atari [6], and navigation robots [35], and multi-agent settings such as traffic light control [36], financial market trading [37], and strategy games such as Go, StarCraft, and Dota are some examples that are developed by DRL.

In DRL, unstructured input data from the state space are given to the network. This input such as pixels rendered on the screen in a video game or images from a camera or the raw sensor stream from a robot can be very large and high-dimensional. In the output, the value of an action is determined for the agent to decide what actions must be performed in the environment to maximize the expected rewards. Since the RL methods are suffered from the curse of dimensions problem. DNNs can find low-dimensional representations (and features) of high-dimensional data automatically. In the following, the subject of DRL for the special scope of multi-agent reinforcement learning will be expressed widely.

3.1 Multi-agent reinforcement learning

In multi-agent reinforcement learning (MARL) sets of independent agents interact with each other to learn how to reach their goals. Large and random state spaces are a common problem in MARL systems with dynamic environments. These challenges include inefficient cooperation between agents, unsuitable coordination between agent decisions, and the effect of state space size on the learning time. In recent years, MARL applications have been used in autonomous driving, traffic light control, and network packet delivery. Communication between agents gathers information about the environment and the status of other agents.

Markov decision process (MDP) is a useful approach for modeling optimal decision-making in stochastic environments such as multi-agent environments but with different representations. The dynamics of the state and the expected rewards change for agents according to the common action and violate the stationarity assumption of MDP. MDP can be completely or partially observable in a multi-agent environment. It also depends on the type of interaction, which can be competitive, collaborative, or mixed. They perform actions sequentially or simultaneously.

Markov game [38], represent a theoretical framework for the study of agents with multiple interactions in a fully observable environment and can be used in competitive, cooperative, and hybrid environments. A Markov game is a set of regular games (matrix games) that agents perform repeatedly in it. Each state of the game can be represented as a matrix representation with the payoff of each joint action. If the agents cooperate with each other; but the actions have decentralized execution, it is shown by a decentralized MDP.

A partially observable Markov game is a multi-agent Markov decision process in which every agent has an individual partial observation of the environment and takes an individual action to receive their own reward. If the agents cooperate to optimize a single reward function according to the joint state and action, then the problem can be modeled as a decentralized partially observable Markov decision process (Dec-POMDP). RL in a multi-agent space is associated with several problems. Partial observability, non-stationarity, computational complexity, and credit allocation are among these problems. In the following, each of these aspects will be discussed.
3.1.1 Partially observable

Based on local observations in a partially observable environment, each agent makes decisions. So, it leads to asymmetric and incomplete information among agents, which makes the learning process difficult. Partially observable working has been studied mainly in situations where a group of agents maximizes team rewards through a common policy. For example, in Dec-POMDP settings, the two main approaches are (1) centralized learning and decentralized execution paradigm, and (2) using communication to exchange information about the environment.

Since in Dec-POMDP the agents partially observe the state and try to maximize the rewards in each step for all agents, the optimal solution for a Dec-POMDP model is considered a challenge. The lack of access to the real state information in the Dec-POMDP leads to the use of the history of observations and actions, which is computationally expensive for solving the Dec-POMDP model. Policy tree by pruning suboptimal trees [39, 40], and a feature-based heuristic search value iteration [41] techniques are used to solve this challenge in Dec-POMDP model.

Also, deep multi-agent reinforcement learning algorithms for Dec-POMDP models are considered an approximate policy solution technique. Different MARL algorithms have been represented to produce decent policies on many challenging dec-POMDP problems [42, 43]. An independent learning approach is used in [43] where a policy solution for each agent is updated solely based on their individual experiences.

3.1.2 Non-stationary

In a multi-agent environment, all agents simultaneously learn, interact and change the environment. As a result, state transitions and rewards are no longer fixed, and agents continue to adapt to the changing policies of other agents. This violates the Markov assumption, which is problematic because most RL algorithms assume a fixed environment to guarantee convergence. One way to deal with non-stationarity is to learn as much as possible about the environment, e.g., through adversary modeling and information exchange between agents [44].

To solve the non-stationarity problem the centralized critic architecture is used. Actor–critic algorithm for this architecture includes two components. The critics’ training is centralized and has access to the observations and actions of all agents, while the actors’ training is decentralized. Since the actor computes the policy, the critic component can be removed during testing, and therefore the approach has fully decentralized execution. If the actions and observations of the opponent during the training are available, the agents do not experience unexpected changes in the dynamics of the environment and it will lead to the stabilization of the process.

The actor-critic algorithm in [42] is used with stochastic policies to evaluate and train agents in the StarCraft game. A single centralized critic is applied for all the agents and a different actor for each agent is used.

Generally, considering non-stationarity in multi-agent systems does not need centralized training approaches. Self-play is another decentralized method that has been explored to manage non-stationarity in MARL problems. This approach trains a neural network, using each agents’ observation as input, by playing it against its current or previous versions to learn policies that can generalize to any opponent.
3.1.3 Computational complexity

As each agent is added, state-action space grows exponentially, which leads to an increase in the time complexity of algorithms in multi-agent environments. Training a DRL model for a single-agent needs significant sources and gets worse for several interaction agents which leads to slow learning.

Reducing the learning complexity for goal-directed problems can be achieved by initializing the Q-values with a good approximative function. In multi-agent problems, good approximations for a big class of problems, namely for goal-directed stochastic games, exists [45]. These games can reflect coordination cooperative robotics.

3.1.4 Assignment of credit

Allocation of credit to agents due to the simultaneous performance of several agents in the environment leads to the difficulty of learning an optimal policy in the environment. The individual contribution of an agent cannot be determined in the joint reward signal [46]. The agent is also able to distinguish whether changes in global reward are due to its actions or those of other agents. One way to solve this problem is to let agents learn based on a local reward. But the agent may easily increase his local reward, which encourages selfish behavior that may reduce overall group performance. Several approaches are discussed which were created to deal with these challenges.

3.2 Interaction methods between multi-agents with GNN architecture

In the most recent research, many MARL methods use GNNs to provide information interactions between agents to complete collaborative tasks and coordinate actions. In general, not extracting enough useful information from neighboring agents is one of the problems of simply aggregation in GNN, which is due to ignoring the topological relationships in the graph.

To solve this problem, Ding et al. [47] presented a method to extract useful information from neighboring agents as much as possible in the graph structure, which has the ability to provide feature representation to complete the cooperation task. For this purpose, mutual information (MI) is applied for measuring the agent topological relationships and the agent features information to maximize the correlation between input feature information of neighbor agents and output high-level hidden feature representations.

A GNN architecture for training decentralized agent policies on the perimeter of a unit circle has been proposed in continuous action spaces [48]. In this approach, multi-agent perimeter defense problems are solved by learning decentralized strategies with GNNs. Local perceptions of the defenders are considered as inputs in the learning framework and finally, the model is trained by an expert policy based on the maximum matching algorithm and returns actions to maximize the number of captures for the defender team.

The proposed framework [49] used GNNs for value function factorization in multi-agent deep reinforcement learning. A complete directed graph is designed by the team of agents as the nodes of the graph, and edge weights are
determined by an attention mechanism. The introduced mixing GNN (GraphMIX) module in this paper is responsible for factorizing the team value function into individual per-agent observation-action value functions, and explicit credit assignment to each agent. The centralized-training-decentralized-execution in GraphMIX give the ability to the agents to make their decisions independently once training is completed.

An attention mechanism in [50] is defined to adjust the weights of the edges during an episode based on the agents’ observation-action histories. To create the factorized state-action value function’s implicit assignment of global reward, additional per-agent loss terms are taken from the output node embeddings of the GNN, that divide the global reward to individual agents explicitly. Neural attention modules have been used in the graph structures [50, 51], for applying attention mechanisms to compute graph edge weights. These techniques are used in sentence translation works for managing associations between structured data [52], and they are generally used in RL [53].

Non-stationery and coordination problems can be solved naturally by centralized learning of joint actions but it is difficult to scale, because of the exponentially grows of the joint action space by the number of agents. To solve this challenge, conditional independencies between agents are exploited by decomposing a global reward function into a sum of agent-local terms. Sparse cooperative Q-learning [54] is a tabular Q-learning algorithm that learns to coordinate the actions of a group of cooperative agents only in the states in which such coordination is necessary, encoding those dependencies in a coordination graph. The use of these methods requires the prior provision of dependencies between agents. To solve this problem, it is assumed that each agent always contributes to the global reward and learns the amount of its contribution in each state.

Coordinating graph formulation is one of the methods for determining the joint action between agents based on the structure of interactions. In [55], Deep Implicit Coordination Graph (DICG) architecture is introduced, which includes two modules, one for obtaining the dynamic coordination graph structure and the other for learning the implicit reasoning about common actions or values. DICG uses the actor-critic structure to improve coordination for multi-agent situations. DICG is assumed that agents can pass messages that encode their observations. The agents use GCN to pass these messages between one another, where the adjacency matrix for the network is learned with self-attention. Here, a new state categorization method has been presented for centralized-training-decentralized-execution. In this method, which is implemented in the StarCraft game, each game agent separates information and observations of itself and its competitors and then leverages GAT to learn the correlation and relationship among the agents.

### 3.3 Different methods for computing value function in MARL

This section describes different methods for calculating the Q-value function for multi-agent environments. In MARL problems, each agent has a local and private observation of its surrounding space that it wants to take action based on that information. A problem that the agent may face with it is the locality of observation and not having complete information about the environment. Another problem is the non-stationarity of the environment because all agents in the environment are learning and show different behaviors during training.
To solve these problems, the simplest method is to use single-agent RL algorithms for each agent and consider other agents as part of the environment. However, the exponential growth of this joint action space becomes difficult with the number of agents. The Independent Q-Learning (IQL) [56] method is based on this logic and has a good efficiency in some multi-agent RL problems, but there is no guarantee of their convergence. In IQL, each agent has a separate action value function based on which it receives the local observation of the agent and then chooses its action based on it. In such environments, RNN can also be used for the history of observation-action.

In another approach, the agents perform learning in a centralized manner and the choice of action is also centralized. This approach is suitable for problems (such as traffic management or traffic light management) that do not require decentralized execution.

The third approach includes centralized training and decentralized execution. In this approach, the agents have access to the state and complete information during the training step, but in some environments, the learned policy must be applied in a decentralized manner, and the agents cannot access the full state in the execution phase. In this method, the purpose of each agent is to perform actions that maximize their utility function (joint value function), but such decentralization can result in sub-optimal actions [55].

Value-based methods like Value Decomposition Networks (VDN) [57], QMIX [58] and actor-critic methods like Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [59] and Counterfactual Multi-Agent (COMA) [60] are some approaches presented to solve these problems with training in a centralized manner and execution in a decentralized way.

In VDN, a linear summation of all action-value functions of all agents is used to determine separate action-value functions for each agent and learn using only a common reward signal. Using a common reward signal, it tries to learn the decomposed value functions for each agent and use it for decentralized execution. QMIX generalized the VDN method and combines the Q-value of different agents in a non-linear way. They use the global state as input to hyper networks to generate weights and biases of the mixing network. The actor-critic architecture is the basis of centralized training and decentralized execution. In this method, they use the full state and additional information available in the training phase of the critical network to generate a richer signal for the actor.

One of the disadvantages of the aforementioned above algorithms is that it does not clearly obtain the underlying structure of cooperation between agents with a graph topology. Some papers try to join MARL with graph learning. For example, a multi-agent deep reinforcement learning based on GCN structure has been presented [61]. Here, the decentralized decision-making is not considered by the agents and only centralized training and centralized execution are investigated for communicating agents with each other during the inference phase several times.

Multi-agent DDPG (MADDPG), generalizes the actor-critic algorithm into a multi-agent policy gradient algorithm where decentralized agents learn a centralized critic based on the observations and actions of all agents. It leads to learned policies that only use local information and observations at execution time. This method does not assume a differentiable model of the environment dynamics or any particular structure of the communication method between agents. It applies not only to cooperative interaction but also to competitive or mixed interaction involving both physical and communicative behavior. The critic is strengthened by additional information about other agents’ policies, while local information is provided just for the actor. After
training completion, only the local actors are used in the execution phase, acting in a decentralized manner.

COMA is a multi-agent policy gradient-based method for cooperative multi-agent systems that uses a centralized critic to estimate Q performance and decentralized actors to optimize agent policies. Also, this method solves the problem of credit assignment using a count. Unlike COMA, which uses a centralized critic for all agents, MADDPG has a concentrated critic for each agent to have different reward functions in competitive environments.

Recent works have been conducted based on MADDPG, R-MADDPG [62] develops the MADDPG algorithm to the semi-observable environment by preserving the history of previous observations in the critic module and by having an iterative actor. M3DDPG [63] includes minimax optimization for powerful policy learning against agents with changing strategies. Actor-Critic with mean field [64] factorizes the Q-value function only by using interaction with neighboring agents based on mean field theory, and the idea of dropping out can be expanded to MADDPG for managing large input space [65].

4. Combination of graph neural networks and reinforcement learning

Recently, combining GNNs with reinforcement learning for graph-structured problems is a powerful tool in modern deep learning [66]. Combinatorial optimization [67], transportation problems [68], and manufacturing and control [69] are interesting applications in these fields.

Figure 3 shows the total structure of the combination of GNNs and DRL. The local observation of agents is encoded by MLP for low-dimensional input or CNN for visual input into the feature vector which is shown in the embedded layer. The attention

![Figure 3](https://via.placeholder.com/150)
network usually represents to define the edge weights as the strength of the connection in the coordination graph between each agent and its neighbors. In the next step, the graph convolution layer is applied to perform message passing and information integration across all agents. Finally, the deep Q-network is used to approximate the Q-value function. By considering the maximum output of the Q-network the next action for the agents is determined.

The embedding layer contains an encoder for \( n \) observations \( o_1, o_2, \ldots, o_n \) of \( n \) agents. The outputs of the encoder include embedding vectors \( E_i \) for \( i = 1 \ldots n \) as follows:

\[
E_i = \text{Encoder}(o_i, \theta_E) \tag{1}
\]

In the local attention Layer, the attention weights for two agents \( i \) and \( j \) in the graph are calculated using embedding vectors as:

\[
At_{ij} = \frac{\exp(\text{Attention}(E_i, E_j, W_a))}{\sum_{k=1}^{n} \exp(\text{Attention}(E_i, E_k, W_a))} \tag{2}
\]

where the attention network is parametrized by the weight matrix \( W_a \).

Message passing and information integration across all agents are expressed in a graph convolution layer as follows:

\[
H^{l+1} = \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{l} W_c^{l} \right) \tag{3}
\]

where \( H^{l} \) is the feature matrix of convolution layer \( l \), \( \tilde{A} = A + I_N \), and \( \tilde{D}_{ii} = \sum_{j} \tilde{A}_{ij} \).

The predicted \( \tilde{Q} \) in Q-network is verified by \( \theta \) parameter. The general objective for each minibatch in the training step is to minimize the loss function as:

\[
L_{\theta} = \frac{1}{b} \sum_t y_t - \tilde{Q}(s_t, a_t, \theta_{\text{predict}}) \tag{4}
\]

where \( b \) is the batch size, and \( y_t = r_{t+1} + \gamma \max_{a_t} Q(s_{t+1}, a_{t+1}, \theta_{\text{target}}) \) in time step \( t \) is the target of \( Q \) value function for state \( s \) and action \( a \) with reward \( r \).

In general, the combination of GNN and DRL can be addressed from two different points of view. From one perspective, GNN is used to advance the formulation and performance of DRL and specifically, when GNN has been used for relational DRL problems. The successful modeling for this relationship can be defined among (1) different agents in a multi-agent deep reinforcement learning (MADRL) framework, and (2) different tasks in multi-task deep reinforcement learning (MTDRL) framework [70].

From another perspective, DRL can be used to progress the performance of GNN. DRL is used to improve the explanatory power of GNN predictions, Neural Architecture Search (NAS) [71], and design adversarial examples for GNN. NAS is the process of automatically searching for the optimal architecture of a particular neural network to solve a problem, which includes finding the number of layers, the number of nodes in the layer, etc. In GraphNAS [72], the RL algorithm helps to search in the graph neural architectures. GraphNAS represents a search space for covering sampling
functions, aggregation functions, and gated functions. To define the architecture of a graph neural network a recurrent network is used to create variable-length strings. Auto-GNN [73] is defined in the predefined search space by RL-based controllers. This architecture is applied in the hidden dimension, attention head, attention function, activation function, and aggregate function.

Identifying the subgraph that can have the most impact on the prediction process in GNN is one of the problems in generating explanations for GNN predictions, and in [74], DRLs are used for this improvement. Here, a DRL-based iterative graph generator is used the most important node for a prediction as a seed node is selected and then adds edges to generate the explanatory sub-graphs.

Learning a sub-graph generation policy with a policy gradient is done by mutual information of predictions and the distribution of predictions according to the explanatory sub-graph. This method achieves better performances from the point of view of the qualitative and quantitative similarity between the generated sub-graphs and the ground truth explanations.

Another application of DRL is to add or remove existing edges during adversarial attacks on GNNs [75, 76]. RLS2V [77] is a framework that uses DRL to learn structural changes in graphs, which is used to develop strategies for adversarial attacks on GNNs. Since GNNs are vulnerable to adversarial attacks that corrupt or poison the data used to train them. Q-learning and structure-to-vector-based attack methodology are learned to modify the graph structure. The purpose of DRL is to perform an attack aimed at evading detection during classification.

4.1 Multi-agent deep reinforcement learning

Multi-agent deep reinforcement learning needs coordination to efficiently solve certain works. Due to the size of joint action spaces, fully centralized control is often infeasible in these problems. The coordination graph-based method allows reasoning about the joint action based on the structure of interactions.

The coordination graph (CG) is introduced by Guestrin et al. [78], where a method for joint value estimation is presented that allows explicit modeling of the locality of interactions and formal reasoning about given joint actions. CG is a way to factorize a complex multi-agent Q-function. Rather than having a single joint Q function which would depend on the joint action of all agents, one could use a hypergraph to decompose this Q-function into a sum of Q functions across the edges, where each edge denotes a much lower dimensional Q function. Then finding the minimizing joint action can be done by passing messages along the edges of the coordination in a hypergraph.

MAGNet [79] represents policies for multi-agent environments based on relevant graphs and message-passing mechanisms. Here, the graphs are static and constructed based on heuristic rules. Multiple agents in the DGN model [80] are shown as nodes of a graph and relationships between them are learned as the observation encoder module in the environment. In the next step, by a convolutional kernel module, a multi-head point generation attention is defined to extract relational features between each agent and its neighbors in the local region. Q network module receives the extracted features of the former step to use them for determining the strategy which ultimately leads to cooperation between agents. In order to create an effective strategy in cooperation between agents, joint training between the encoder and Q network is done sequentially. This paper proves that GCN increases strongly cooperation among agents. This model is investigated in a grid-world platform MAgent.
Inspired by this idea [80], a model is presented in [81] that controls the connected autonomous vehicles (CAVs) as multi-agents by GNN and RL for cooperation between them. Information transfer for connected autonomous vehicles attains through the onboard sensors of nearby human-driven vehicles (HDVs) as local information and also from other connected autonomous vehicles the global information is obtained via connectivity channels. This information helps to define the graph structure. Within the local network, information passes from HDVs to CAVs. From the global network, all the CAVs can share knowledge including locally sensed information and their own information. Here, the environment contains a variable number of agents and makes a dynamic length output that matches with CAVs driving operations. Due to the variable number of agents, it is difficult to use joint training for each agent with its distinct Q network. Also, joint training is not scalable because by increasing the number of agents, the number of parameters for distinct Q networks will grow exponentially. One efficient method for solving these challenges is to apply a shared centralized Q network for all agents to determine their actions. Using the combination of GCN and deep Q network can have collaborative and safe controlling for lane-changing decisions in different traffic.

4.2 Multi-task deep reinforcement learning

MTDRL prepares a learning framework for coordinating and exploiting commonalities between multiple tasks in order to learn data efficiency, and robustness policies with improved efficiency, and generalization. Compatible state-action spaces are the main assumption in a MTDRL process such as the same dimensions of states and actions across multiple tasks. This issue is supported by GNNs due to capable of processing graphs with arbitrary sizes.

One of the applications of GNN in a MTDRL is in continuous control environments that use the features of each element of the MuJoCo agent to construct input graphs [82]. Each actuator has obtained the information from local sensors. A shared modular policy is defined as a global policy for each agent’s actuators. Each limb of the MuJoCo agent is considered as a state with features containing positions, rotation, velocity, etc. that implements its independent policy to optimize joint reward function.

A framework in [83] is proposed to learn a job-shop scheduling problem (JSSP) by GNN and RL. The GNN section contains the creation of a graph from spatial features of the element of the job-shop problem and the RL section considers it as sequential decision-making by proximal policy optimization method (PPO) as a scheduling process.

5. Conclusion

In this survey, we tried to summarize about GNNs and RL and their relations. We had an overview of the challenges inherent in graph neural networks and multi-agent environments. Since, learning in collaborative multi-agent environments with dynamic, non-deterministic, and large state space has become a very important challenge in applications. Among these challenges, we can mention the effect of the size of the state space on the duration of learning, as well as the inefficient cooperation and the lack of proper coordination in decision-making between the agents. Also, when using reinforcement learning algorithms with the graph structure, the models will face
challenges such as the difficulty of determining the appropriate learning goal and the long convergence time caused by trial and error-based learning. So, the integration of these methods leads to more realistic scenarios and more effective solutions to real-world problems. Researchers in this field have a significant impact on the progress of the combination of GNNs and DRL by providing newer models and architectures.

Author details
Fatemeh Fathinezhad1*, Peyman Adibi1, Bijan Shoushtarian1 and Jocelyn Chanussot2

1 Faculty of Computer Engineering, Artificial Intelligence Department, University of Isfahan, Isfahan, Iran
2 GIPSA-Lab, CNRS, Grenoble INP, University of Grenoble Alpes, Grenoble, France

*Address all correspondence to: fateme.fathinezhad@eng.ui.ac.ir

© 2023 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.
References


[16] Veselkov K et al. Hyperfoods: Machine intelligent mapping of cancer-
beating molecules in foods. Scientific Reports. 2019;9(1):1-12


Deep Learning and Reinforcement Learning


