Title: Graph Reinforcement Learning for Improving Smart Grid Services

Author: António Oliveira

Supervision: António Costa

Co-Supervision: Rosaldo Rossetti

Date: January 8, 2024

1 List of Acronyms

IT Information Technology

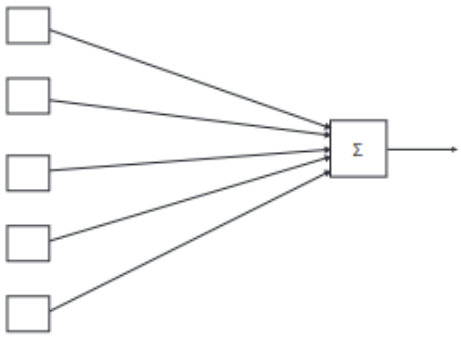

Figure 1: The Perceptron [1]

2 Background Knowledge

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of machine learning algorithms based on the neural process of biological learning. The simplest form of an ANN is a Multilayer Perceptron (MLP), also called a feedforward network, whose main objective is to approximate to function f that models relationships between input data x and output data y of considerate complexity [1, 2]. It defines a mapping $y = f(x;\theta)$ and learns the best composition of parameters θ to approximate it to the unknown model. The MLP serves as a fundamental part of developing the other more complex types of neural networks [1].

2.1.1 Feedforward Neural Networks

The main building block of a MLP is the *Perceptron*, pictured in figure 1, a simple computational model initially designed as a binary classificator that mimics biological neurons' behaviour [1]. A neuron might have many inputs x and has a single output y. It contains a vector of *weights* $w = (w_1...w_m)$, each associated with a single input, and a special weight b called the *bias*. In this context, a perceptron defines a computational operation formulated as equation 1 portrays [1].

$$f(x) = \begin{cases} 1 & \text{if } b + \mathbf{w} \cdot \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Functions that compute $b + \mathbf{w} \cdot \mathbf{x} > 0$ are called *linear units* and are identified with Σ [1, 2]. An activation function g was introduced to enable the output of non-linear data. The default recommendation is the *Rectified Linear Unit* ($ReLU$), with the Logistic curve (sigmoid) also being very common [2].

Feedforward networks are composed of an input layer formed by the vector of input values, an arbitrary number of hidden layers and an output layer, which is the last layer of neurons [1]. The greater the amount of layers the higher the *depth* of the network [1, 2].

On its own, the model amounts only to a complex function. Still, with real-world correspondence between input values and associated outputs, a feedforward network can be trained to approximate the unknown function of the environment. In more concrete terms, this involves

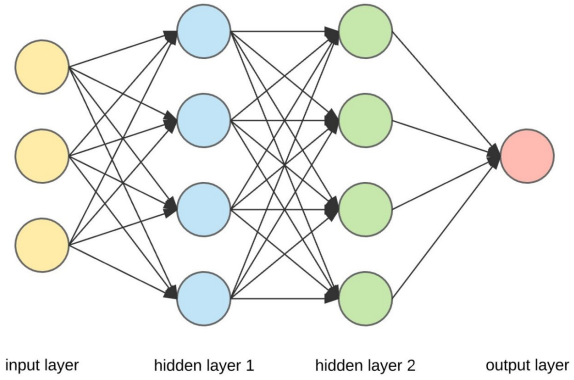

Figure 2: Feedforward Neural Network [3]

updating all of the different weight and bias values of each neuron to achieve an output as close as possible to the real or desired value or minimize the total loss, which indicates how distant the network model is to the real function to approximate [1, 2]. *Loss functions* are used to calculate this value.

2.2 Reinforcement Learning

Reinforcement Learning (RL) consists of a field and a class of machine learning algorithms that study how to learn to take good sequences of actions to achieve a goal associated with a maximizing received numerical reward [4]. The main objective is to maximize the received cumulative reward by trying between the available actions and discovering which ones yield the most reward [5]. This sequential decision-making process becomes more complex when a delayed reward is considered, given that an action with immediate reward may not always reflect the delayed consequences of that decision [5]. It's also the learner's job to consider this during the learning process. These concepts of *delayed reward* and *trial-and-error search* make up the most important characteristics of Reinforcement Learning [5]. The classic formalisation of this problem is the Markov Decision Process (MDP) through defining the agent-environment interaction process, explained in the following subsection 2.2.1.

A major challenge in this machine learning paradigm is the trade-off between *exploring* new unknown actions and *exploiting* the already known "good" actions [5]. To choose the sequence of actions that return the highest reward, the agent must choose actions it found effective in similar past situations or *exploit* what it learned from experience. Furthermore, given that the agent may not know the action-reward mappings initially, it has to *explore* possible actions that were not selected previously or may initially seem to yield a low reward to compute accurate reward estimates. The main problem is that neither exploitation nor exploration can be favoured exclusively without failing at the task [5]. Additionally, an agent's environment is uncertain, and changes in the environment's dynamics may also involve re-estimating action rewards.

In conclusion, RL techniques enable the implementation of sequential decision-making agents that seek to maximize a reward signal analogous to an explicit (complex) goal. The agents need to balance between actions that yield a reward on posterior time steps and actions that produce immediate rewards. In addition, these agents are also faced with the task of balancing the exploitation of information from past experiences and the exploration of new decision paths that could potentially return a higher reward down the road [5].

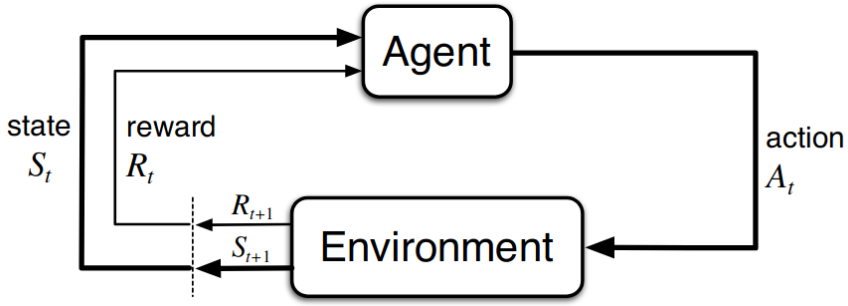

Figure 3: Interaction between agent and environment in an MDP [5]

2.2.1 Markov Decision Process

Markov Decision Processs (MDPs) are a classical formalization of a sequential decision-making process, constituting the mathematical definition of the RL problem [5, 6]. Beyond estimating potential rewards for the available actions, the problem defined by MDPs involves learning which actions are optimal in specific situations, i.e. learning a mapping between states of the environment and actions [5].

The central component of MDPs is the agent, which acts as a decision-maker and learns from interactions with the environment it's inserted. In a continuous process, the agent takes actions that affect the environment's state, which in turn presents new situations [5]. The environment also responds with the reward signals which the agent aims to maximize over time through its decision process.

Formally, the agent-environment interactions, as figure 3 entails, occur in a sequence of discrete time steps t, where at each step, the agent receives a representation of the state of the environment $S_t \in \mathcal{S}$ which is used to select an appropriate action $A_t \in \mathcal{A}(s)$, where $\mathcal{S}$ is the set of possible states called the *state space* and $\mathcal{A}(s)$ is the set of available actions for state s [5, 6]. In the next step, the agent receives, as a consequence of its decision, a numerical reward signal $R_{t+1} \in \mathcal{R} \subset R$ and is faced with a new state S_{t+1} [5]. Ultimately, the MDP agent follows a logical sequence that occurs as equation 2 states. The collection of a state S_t, action taken A_{t+1}, reward R_{t+1} received and next state S_{t+1} constitutes an *experience tuple* [6].

$$S_0, A_0, R_1, S_1, A_2, R_2, S_2, A_2, R_3, \ldots \tag{2}$$

In addition, when the set of possible actions, states and rewards ($\mathcal{A}$, $\mathcal{S}$ and $\mathcal{R}$) are finite, the MDP is said to be *finite* [5]. This results in S_t and R_t having well-defined discrete probability distributions in function of the preceding state and chosen action [5]. Therefore, the probability of receiving a particular reward and state given the previous state and selected action, which characterizes a finite MPD's dynamics, may be characterized by function p defined in equation 3

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{3}$$

For all $s, s' \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}(s)$, where $\doteq$ denotes a mathematical formal defintion. This encompasses the assumption that the probability of each possible state, S_t, and reward, R_t, pair is only dependent on the preceding state, S_{t-1}, and action taken, A_{t-1} [5]. Instead of observing this as a restriction on the decision process, it's more convenient to view it as a

constraint on the state variable, considering that it must contain all the necessary information from experience to make a valuable decision in the immediate step. If this condition is satisfied, the state is declared to have the *markov property* [5].

From function p in equation 3, the state-transtion probabilities, also called the *transition function*, can be computed as described by equation 4 [5, 6].

$$p(s'|s,a) \doteq Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a) \tag{4}$$

In addition, the expected rewards can be calculated for state-action pairs (equation 5) or state-action-next-action triples (equation 6) [5, 6].

$$r(s,a) \doteq E[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a) \tag{5}$$

$$r(s,a,s') \doteq E[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r\in\mathcal{R}} r\frac{p(s', r|s, a)}{p(s'|s, a)} \tag{6}$$

2.2.2 Rewards and Returns

As stated in the previous subsections, the main goal of a RL agent defined by the numeric reward signal, $R_t \in R$, it receives from the environment [5]. In this context, the agent's objective is to maximize the total reward it receives, considering not only immediate but also the cumulative reward over time. In the ubiquitous work of [5], the *reward hypothesis* is stated as follows:

> That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward). [5]

This also entails that the process of reward maximization from the agent has to be closely tied to it achieving its defined goals in a practical sense. Otherwise, the agent will fail at fulfilling the desired objectives [5].

Formally, the goal of an RL agent can be defined by the maximization of the cumulative reward received of time called the *expected return*, G_t [5].

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{7}$$

T describes the final time step. This definition can be applied in domains with a natural notion of a terminal state or final time step. In these cases, the agent-environment interaction process can be broken into logically independent subsequences called *episodes* [5]. Each episode ends in a special state, called the terminal state, restarting a new sequence of states and actions completely independent from the previous episode [5]. In this context, episodes can be considered to end in the same terminal state, with different accumulated rewards for the different outcomes [5].

In contrast, there are situations where the decision-making process doesn't divide itself into logically identifiable episodes but goes on indefinitely. In this case, $T = \infty$ and according to equation 7, the expected return the agent aims to maximize would be infinite [5]. In this manner, another concept is added in the expected return definition called the *discount rate*,

γ where $0 \leq \gamma \leq 1$, representing how strongly the agent should account for future rewards in the expected return calculations, as equation 8 [5].

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{8}$$

From this equation, we can compute the expected discounted return on a given time step t in the function of the immediate reward signal received and the expected return for the next time step $t+1$, which eases the job of calculating expected returns for reward sequences [5]. This is entailed by equation 9.

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{9}$$

In this manner, a MDP can be defined by a tuple with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function p, a reward function r and a discount factor γ, as equation 10 portrays [4].

$$M = (\mathcal{S}, \mathcal{A}, p, r, \gamma) \tag{10}$$

2.2.3 Policies and Value Functions

RL techniques typically involve the estimation of what is understood as *value functions*, functions that estimate the expected return based on the current state value or state-action pair. This characterizes how good is for an agent to be in a specific state or to take an action in a specific state, respectively, using the expected return to characterize the overall *goodness* of these scenarios [5, 6]. These functions are tied to a specific way of determining the action in a given state. Formally, this is defined as a *policy* π, that defines the probability $\pi(a|s)$ of taking action a in state s [5]. In this context, the *state-value function*, $v_\pi(s)$ and *action-value functions* $q_\pi(s, a)$ for policy π can be defined by equations 11 and 14, respectively [5].

$$v_\pi(s) \doteq E_\pi[G_t|S_t = s], \forall s \in \mathcal{S} \tag{11}$$

$$q_\pi(s, a) \doteq E_\pi[G_t|S_t = s, A_t = a] \tag{12}$$

The utility of such functions rely on the possibility of estimating them with regard to past experience of the agent [5]. A fundamental property of value functions is that it can, as was the case with the expected return (equation 9), satisfy recursive relationships with the next immediate value as equation 13 entails [5]. This equation is called the *Bellman equation for* v_π, which characterizes the relationship between the value of current and subsequent states.

$$v_\pi \doteq \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{13}$$

$$q_\pi(s, a) \doteq \sum E_\pi[G_t|S_t = s, A_t = a] \tag{14}$$

2.2.4 Types of RL

Regarding RL algorithms, they can be divided into model-free and model-based techniques [7]. These categories are distinguished by whether an agent uses a provided or learned *model* of the set of transition and reward functions, another optional element of RL techniques

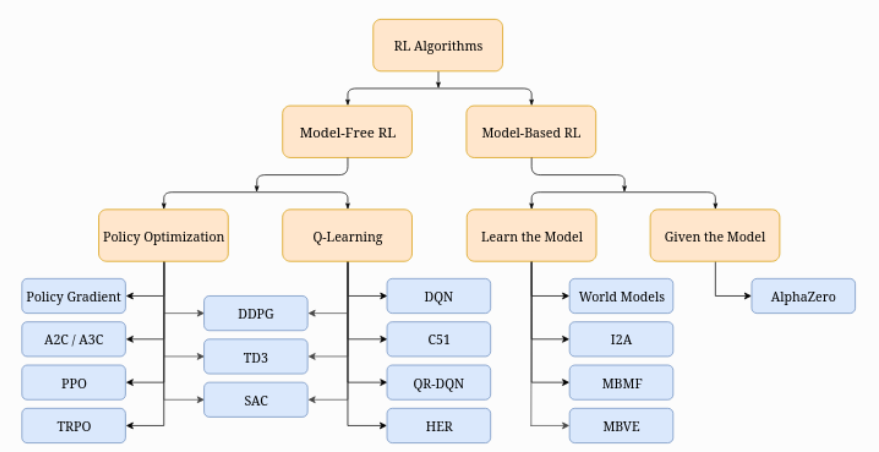

Figure 4: Taxonomy of algorithms in modern RL [7]

[6, 7]. In the positive case, the method is said to be model-based, otherwise, it's model-free. Having an accurate model of the environment allows the RL agent to focus on planning ahead by calculating future scenarios and creating policies based on the results of the planning process. An example of a famous system of this kind is AlphaZero [8]. However, in most cases, agents can't access a ground-truth model of the environment, leaving only the scenario where an agent learns a model purely from experience. This creates several challenges, the most prominent of which relies on the fact that the model, in most times, doesn't fully capture the environment's transition dynamics, equipping it with bias in relation to the actual dynamics. With this, learning how to generalise the model to real-world environments so that the bias is not over-exploited becomes a very complex task [7]. Model-free algorithms can be also further divided into Q-learning and Policy Aproximmation techniques.

Furthermore, algorithms can also be subdivided into on-policy and off-policy methods. [6] On-policy algorithms evaluate and improve a single policy used to determine the agent's behaviour [6]. The methods under the policy optimization category such as A2C and A3C [9] or the Proximal Policy Optimization (PPO) [10] almost always fall into this label. In contrast, off-policy algorithms learn how to improve a different target policy based on the results that arise from the policy used to determine the system behaviour initially [6]. Such approaches include Q-Learning algorithms such as Deep Q-Networks (DQNs) [11, 7]. Deep Deterministic Policy Gradient (DDPG) [12] combines policy optimization with q-learning, consisting of an off-policy method that learns both a q-function and a policy. DDPG constitutes the adaption of q-learning methods to continuous action spaces [7]. Another example of an off-policy algorithm is the Soft Actor-Critic (SAC) [13] method, which bridges stochastic policy optimization with the DDPG approach and has entropy regularization as one of its central features, which translates into training a policy that maximizes the expected return and entropy, a measure of randomness in the policy [7].

Lastly, with the advent of deep learning becoming one of the most ubiquitous techniques in machine learning, RL algorithms have evolved beyond the traditional tabular methods [6]. Traditional RL has evolved to Deep Reinforcement Learning (DRL), which studies how to use deep neural networks in RL problems to leverage their generalization abilities for solving more complex problems.

2.3 Graph Representation Learning

Several objects and problems can be naturally expressed in the real world using graphs, such as social networks, power grids, transportation networks, recommendation systems or drug discovery. The usefulness of such representations is tied to how they instinctively represent the complex relationships between objects. However, graph data is often very sparse and complex, and their sophisticated structure is difficult to deal with [14, 15].

Furthermore, the performance of machine learning models strongly relies not only on their design but also on good representations of the underlying information [14]. Ineffective representations, on the one hand, can lack important graph features and, on the other, can carry vast amounts of redundant information, affecting the algorithms' performance in leveraging the data for different analytical tasks [14, 16].

In this context, **Graph Representation Learning** studies how to learn the underlying features of graphs to extract a minimal but sufficient representation of the graph attributes and structure [17, 15, 18]. Currently, the improvements in deep learning allow representation learning techniques consisting of the composition of multiple non-linear transformations that yield more abstract and, ultimately, more useful representations of graph data [18].

2.4 Graph Neural Networks

In the present, deep learning and ANN have become one of the most prominent approaches in Artificial Intelligence research [18]. Approaches such as recurrent neural networks and convolutional networks have achieved remarkable results on Euclidean data, such as images or sequence data, such as text and signals [19]. Furthermore, techniques regarding deep learning applied to graphs have also experienced rising popularity among the research community, more specifically **Graph Neural Networks (GNNs)** that became the most successful learning models for graph-related tasks across many application domains [18, 19].

The main objective of GNNs is to update node representations with representations from their neighbourhood iteratively [20]. Starting at the first representation $H^0 = X$, each layer encompasses two important functions:

- **Aggregate**, in each node, the information from their neighbours
- **Combine** the aggregated information with the current node representations

The general framework of GNNs, outlined in [20], can be defined mathematically as follows:

Initialization: $H^0 = X$
For $k = 1, 2, \ldots, K$

$$a_v^k = \text{AGGREGATE}^k\{H_u^{k-1} : u \in N(v)\}$$
$$H_v^k = \text{COMBINE}^k\{H_u^{k-1}, a_v^k\}$$

Where $N(v)$ is the set of neighbours for the v-th node. The node representations H^K in the last layer can be treated as the final representations, which sequentially can be used for other downstream tasks [20].

2.4.1 Graph Convolutional Network

A **Graph Convolutional Network (GCN)** [21] is a popular architecture of GNNs praised by its simplicity and effectiveness in a variety of tasks [14, 20]. In this model, the node representations in each layer are updated according to the following convolutional operation:

$$H^{k+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^k W^k) \tag{15}$$

$\tilde{A} = A + I$ - Adjacency Matrix with self-connections
$I \in R^{N \times N}$ - Identity Matrix
$\tilde{D}$ - Diagonal Matrix, with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
σ - Activation Function
$W^k \in R^{F \times F'}$ - Laywise linear transformation matrix (F and F' are the dimensions of node representations in the k-th and $(k+1)$ layer, respectively)

$W^k \in R^{F \times F'}$ is a layerwise linear transformation matrix that is trained during optimization [20]. The previous equation 15 can be dissected further to understand the *AGGREGATE* and *COMBINE* function definitions in a GCN [20]. For a node i, the representation updating equation can be reformulated as:

$$H_i^k = \sigma(\sum_{j \in \{N(i) \cup i\}} \frac{\tilde{A}_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k) \tag{16}$$

$$H_i^k = \sigma(\sum_{j \in N(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k) + \frac{1}{\tilde{D}_i} H^{k-1} W^k) \tag{17}$$

In the second equation, the *AGGREGATE* function can be observed as the weighted average of the neighbour node representations [20]. The weight of neighbour j is defined by the weight of the edge (i, j), more concretely, A_{ij} normalized by the degrees of the two nodes [20]. The *COMBINE* function consists of the summation of the aggregated information and the node representation itself, where the representation is normalized by its own degree [20].

Spectral Graph Convolutions

Regarding the connection between GCNs an spectral filters defined on graphs, spectral convolutions can be defined as the multiplication of a node-wise signal $x \in R^N$ with a convolutional filter $g_\theta = diag(\theta)$ in the *Fourier domain* [14, 20], formally:

$$g_\theta \star \mathrm{x} = U_{g_\theta} U^T \mathrm{x} \tag{18}$$

$\theta \in R^N$ - Filter parameter
U - Matrix of eigenvectors of the normalized graph Laplacian Matrix $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

The eigendecomposition of the Laplacian matrix can also be defined by $L = U\Lambda U^T$ with Λ serving as the diagonal matrix of eigenvalues and $U^T\mathrm{x}$ is the graph Fourier transform of the input signal x [20]. In a practical context, g_θ is the function of eigenvalues of the normalized graph Laplacian matrix L, that is $g^\theta(\Lambda)$ [14, 20]. Computing this is a problem of quadratic complexity to the number of nodes N, something that can be circumvented by approximating $g_\theta(\Lambda)$ with a truncated expansion of Chebyshev polynomials $T_k(x)$ up to K-th order [14, 20]:

$$g_{\theta'}(\Lambda) = \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \tag{19}$$

$\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - \mathrm{I}$
$\lambda_{\max}$ - Largest eigenvalue of L
$\theta' \in R^N$ - Vector of Chebyshev coefficients
$T_k(x)$ - Chebyshev polynomials
$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$

By combining this with the previous equation, the first can be reformulated as:

$$g_\theta \star \mathrm{x} = \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})\mathrm{x} \tag{20}$$

$\tilde{L} = \frac{2}{\lambda_{\max}}L - I$

From this equation, it can be observed that each node depends only on the information inside the K-th order neighbourhood and with this reformulation, the computation of the equation is reduced to $O(|\xi|)$, linear to the number of edges ξ in the original graph G.

To build a neural network with graph convolutions, it's sufficient to stack multiple layers defined according to the previous equation, each followed by a nonlinear transformation. However, the authors of GCN [21] proposed limiting the convolution number to $K = 1$ at each layer instead of limiting it to the explicit parametrization by the Chebyshev polynomials. This way, each level only defines a linear function over the Laplacian Matrix L, maintaining the possibility of handling complex convolution filter functions on graphs by stacking multiple layers [21, 20]. This means the model can alleviate the overfitting of local neighbourhood structures for graphs whose node degree distribution has a high variance [21, 20].

At each layer, it can further considered $\lambda_{\max} \approx 2$, which the neural network parameters could accommodate during training [20]. With these simplifications, the equation is transformed into:

$$g_{\theta'} \star \mathrm{x} \approx \theta'_0\mathrm{x} + \theta'_1\mathrm{x}(L - I_N)\mathrm{x} = \theta'_0\mathrm{x} - \theta'_1 D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \tag{21}$$

θ'_0 and θ'_1 - Free parameters that can be shared over the entire graph

The number of parameters can, in practice, be further reduced, minimising overfitting and minimising the number of operations per layer as well [20] as equation 22 entails.

$$g_\theta \star \mathrm{x} \approx \theta(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})\mathrm{x} \tag{22}$$

$\theta = \theta'_0 = -\theta'_1$

One potential problem is the $I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ matrix whose eigenvalues fall in the $[0, 2]$ interval. In a deep GCN, the repeated utilization of the above function often leads to an exploding or vanishing gradient, translating into numerical instabilities [14, 20]. In this context, the matrix can be further renormalized by converting $I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ into $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ [14, 20]. In this case, only the scenario where there is one feature channel and one filter is considered which can be then generalized to an input signal with C channels $X \in R^{N\times C}$ and F filters (or hidden units) [14, 20]:

$$H = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW \tag{23}$$

$W \in R^{C \times F}$ - Matrix of filter parameters
H - Convolved Signal Matrix

2.4.2 Graph Attention Network

Graph Attention Network (GAT) [22] is another type of GNNs that focuses on leveraging an attention mechanism to learn the importance of a node's neighbours. In contrast, the GCN uses edge weight as importance, which may not always represent the true strength between two nodes [20, 22].

The Graph Attention Layer defines the process of transferring the hidden node representations at layer $k-1$ to the next node presentations at k. To ensure that sufficient expressive power is attained to allow the transformation of the lower-level node representations to higher-level ones, a linear transformation $W \in R^{F \times F'}$ is applied to every node, followed by the self-attention mechanism, which measures the attention coefficients for any pair of nodes through a shared attentional mechanism $a : R^{F'} \times R^{F'} \to R$ [20, 22]. In this context, relationship strength e_{ij} between two nodes i and j can be calculated by:

$$e_{ij} = a(WH_i^{k-1}, WH_j^{k-1}) \tag{24}$$

$H_i^{k-1} \in R^{N \times F'}$ - Column-wise vector representation of node i at layer $k-1$ (N is the number of nodes and F the number of features per node)
$W \in R^{F \times F'}$ - Shared linear transformation
$a : R^{F'} \times R^{F'} \to R$ - Attentional Mechanism
e_{ij} - Relationship Strength between nodes i and j

Theoretically, each node can attend to every other node on the graph, although it would ignore the graph's topological information in the process. A more reasonable solution is to only attend nodes in the neighbourhood [22, 20]. In practice, only first-order node neighbours are used, including the node itself, and to make the attention coefficients comparable across the various nodes, they are normalized with a *softmax* function:

$$\alpha_{ij} = \text{softmax}_j(\{e_{ij}\}) = \frac{exp(e_{ij})}{\sum_{l \in N(i)} exp(e_{il})}$$

Fundamentally, α_{ij} defines a multinomial distribution over the neighbours of node i, which can also be interpreted as a transition probability from node i to each node in its neighbourhood [20]. In the original work [22], the attention mechanism is defined as a single-layer Feedforward Neural Network that includes a linear transformation with weigh vector $W_2 \in R^{1 \times 2F'}$ and a LeakyReLU nonlinear activation function with a negative input slope $\alpha = 0.2$ [20, 22]. More formally, the attention coefficients are calculated as follows:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(W_2[WH_i^{k-1} || WH_j^{k-1}]))}{\sum_{l \in N(i)} \exp(\text{LeakyReLU}(W_2[WH_i^{k-1} || WH_l^{k-1}]))} \tag{25}$$

$||$ - Vector concatenation operation

The novel node representation is a linear composition of the neighbouring representations with weights determined by the attention coefficients [22, 20], formally:

$$H_i^k = \sigma(\sum_{j \in N(i)} \alpha_{ij} WH_j^{k-1}) \tag{26}$$

Multi-head Attention

Multi-head attention can be used instead of self-attention, determining a different similarity function over the nodes. An independent node representation can be obtained for each attention head according to the equation bellow [22, 20]. The final representation is a concatenation of the node representations learned by different heads, formally:

$$H_i^k = \Big\|_{t=1}^{T} \sigma\Big(\sum_{j \in N(i)} \alpha_{ij}^t W^t H_j^{k-1}\Big)$$

T - Number of attention heads
α_{ij}^t - attention coefficient computed from the t-th attention head
W^t - Linear transformation matrix of the t-th attention head

Lastly, the author also mentions that other pooling techniques can be used in the final layer for combining the node representations from different heads, for example, the average node representations from different attention heads [22, 20].

$$H_i^k = \sigma\Big(\frac{1}{T}\sum_{t=1}^{T}\sum_{j \in N(i)} \alpha_{ij}^t W^t H_j^{k-1}\Big) \tag{27}$$

2.5 Smart Grid Services

Given the global ecological emergency and the increasing energetic crisis, there is a necessity for advancements in energy distribution and transmission systems now more than ever. To fulfil the need for energy sustainability, traditional centralized distribution grids must be adapted to, on the one hand, accommodate the rise of distributed renewable energy sources in corporate and domestic consumers and, on the other, to make more efficient and reliable distribution of energy resources [23, 24].

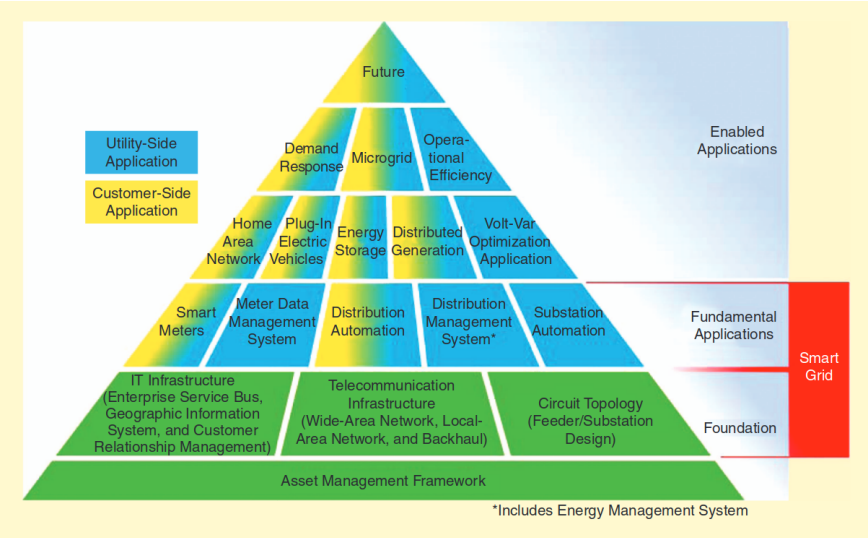

Figure 5: Smart Grid Pyramid [23]

The *Smart Grid* or the *Smart Power Grid* conceptualizes this modernization of the electricity network by leveraging the technological advancements in information technology and

communication science to create intelligent systems that manage and monitor the distributed generation of energy [25, 23]. Figure 5 describes the smart grid pyramid with asset management at its base. On this base, the foundation of the smart grid is laid out by the circuit topology, IT systems and telecommunications infrastructure, the basic ingredients for the emergence of the fundamental applications such as smart meters and distribution automation [23]. In turn, these serve as building blocks for creating more intelligent systems that leverage upper-layer applications, enabling the true smart grid capabilities [23].

3 Related Works

Graph Reinforcement Learning (GRL) or Reinforcement Learning on Graphs is a relatively new area in the broader field of machine learning. GRL techniques have shown significant progress in solving problems with underlying graph-based representations such as power grid management [26, 27], smart transport [28, 29] or task offloading [30, 31]. In this work, the main focus lies on studying the development of GRL techniques and subsequent application to smart grid services such as dynamic economic energy dispatch systems [32, 33], residential electricity behavior identification and energy management [27], or Volt-VAR regulation [34].

Research on this topic has significantly increased in the last few years with the improvements of DRL techniques and the developments in GNNs in the mid-2010s [21, 22, 35, 36]. GNNs became the state-of-the-art for solving numerous data mining tasks involving graph-structured data, excelling at classification, link prediction and representation learning [37, 38]. This advancement brought more sophisticated RL applications on graphs and the surge of a new field studying how to combine the improvements of graph mining and reinforcement learning techniques.

In this context, this literature review is divided into the two main approaches in GRL, which compromise the popular GCN architecture that has been widely researched or leveraging the rising and promising GAT architecture. Lastly, other relevant approaches that use different architectures are also listed.

3.0.1 Plain GCN-Based GRL Techniques

A common approach in Graph Reinforcement Learning model implementation is the use of graph convolutions with the GCNs architecture for leveraging graph-based structures to extract and aggregate the essential features of data in hand and improve the performance of RL agents in those environments. The techniques listed in this subsection constitute approaches that integrate a GCN with RL algorithms.

[26] implements a GRL system to improve the decision quality of economic dispatch under high penetration of distributed energy generations. To accomplish this, a SAC system is employed with the main objective of finding the optimal action policy for minimizing generation cost with the appropriate reliability concerns. This problem is represented by an undirected graph with nodes describing the power grid elements with their respective attributes and edges describing the underlying energy connections between those units. To extract the structural features of the graph, this work implements a full connected layer to perform feature transformation with a two-layer GCN followed by three full connected layers for the non-linear mapping of state-to-action policy in both actor and critic modules. [32] develops a similar approach, with both concluding that it significantly reduces learning time for achieving better training results in comparison to plain SAC and showing significant improvement on economy and flexibility of the system on more complex and sparse state graphs.

The use of GCNs enables the system to adapt to changes in the state space by leveraging the network's generalization ability.

In [39] a three-layer GCN is used to extract node feature and graph topology information and is integrated into a Rainbow-based [40] DRL algorithm for electric vehicle charging guidance. In this article, the testing results show promising performance in reducing operation costs for electric vehicle users, portraying a model with good generalization ability in untrained scenarios.

Another interesting implementation of this approach is [41], which studies and compares different solutions for optimizing autonomous exploration under uncertain environments. It analyzes combinations of a single agent Deep Q-Network (DQN) and Advantageous Actor-Critic (A2C) with Graph Convolutional Networks, Gated Graph Recurrent Networks and Graph U-Nets. The paper reports that the GCN-DQN was the model that achieved the highest reward during policy training, followed by the GGNN-A2C model, although in the end, it concludes that the second showed improved scalability in relation to the first model. In [27] a DQN with a GCN is also used for residential electricity behaviour identification and energy management.

Table 1: GCN-Based GRL Techniques

Reference	**DRL Algorithm**	**Application Domain**
[26], [32]	GCN-SAC	Dynamic economic energy dispatch
[42]	GCN-SAC	Multi-access Edge Computing
[43]	GCN-A3C	Automatic Virtual Network Embeddings
[44]	GCN-DDPG	Automatic transistor sizing
[41]	GCN-DQN	Autonomous Exploration under uncertainty
[27]	GCN-DQN	Residential electricity behavior identification and energy management
[39]	GCN - Modified Rainbow	Electrical Vehicle Charging Guidance
[45]	GCN-MDP	Interpret GNNs at model-level
[46]	GCN-DQ	Task Offloading in Edge Computing

3.0.2 Attention-based GRL Techniques

Another effective approach in extracting relevant topology and graph features relies on using attention mechanisms to weigh different nodes' contributions dynamically. While this encompasses techniques that use the GAT architecture, which is a GNN design with the attention mechanism at its core, various scholars propose GCN approaches integrated with attention mechanisms such as [47] and [48]. [33] proposes a DDPG-based algorithm improved with a GAT block with three graph Attention Layers for extracting and learning the topology infor-

mation for achieve real-time optimal scheduling for Active Distribution Networks (ADNs). This paper compares the obtained test results against a GCN-DDPG model and shows increased performance over the GCN method in reducing cost and power loss. Beyond this, the work demonstrates that the GAT's attention mechanism enables the algorithm to focus on more important nodes and improve the signal-to-noise ratio compared to its GCN counterpart. [49] and propose a multi-agent approach to the same domain but more focused on voltage regulation with a multi-agent SAC instead of a single-agent DDPG algorithm.

In [28], another model for the electric vehicle charging guidance is proposed, consisting of a bi-level approach of a Rainbow-based algorithm with a GAT block. The upper level focuses on the decision-making process regarding charging, while the lower level handles routing. The proposed model proved to be more effective than a shortest distance path-based [50] and a DRL-based [51] approach. It suggests that in a future direction, developing GNNs directly embedded into the RL framework might further improve the model's robustness and scalability. [52] develops a similar approach with a Double-prioritized DQN for the same application domain. In [47] and [48], the sequential distribution system restoration problem is addressed with a multi-agent RL algorithm equipped with a GCN with an attention mechanism. In the first case, multi-head attention is used as the convolution kernel for the GCN with a DQN algorithm. In the second, self-attention is used for improving the centralized training of the used multi-agent actor-critic algorithm, more concretely, by embedding it in the critic networks. At the same time, the GCN is integrated into the actor networks for extracting the graph features. Both solutions proved more efficient than traditional RL techniques, with the first highlighting its solution generalizability and the second showing increased scalability facing the non-GRL techniques.

Table 2: Attention-Based GRL Techniques

Reference	**DRL Algorithm**	**Application Domain**
[33]	GAT-DDPG	Optimal Scheduling for ADNs
[49]	GAT-MASAC	Multi-agent Voltage Regulation
[28]	GAT-Modified Rainbow	Electric Vehicle Charging Guidance
[52]	GAT-DQN	Electric Vehicle Charging Guidance
[47]	GCN-DQN	Multi-agent Sequential Distribution System Restoration
[48]	GCN-MAAC	Multi-agent Service Restoration

3.0.3 Other Approaches

This subsection includes GRL approaches that combine of other GNNs architectures with RL algorithms. In [53], a GraphSAGE network is used with a Deep Dueling Double Q-Network (D3QN) for emergency control of Undervoltage load shedding for power systems with various topologies. The author presents promising results for the GraphSAGE-D3QN model compared to a GCN-D3QN, achieving higher cumulative reward and faster voltage recovery speed, although it required longer decision times. The proposed model performed excellently in the application domain and successfully generalised the learned knowledge to new topology variation scenarios.

[54] focused on solving the Job shop scheduling problem through a priority dispatching rule with a Graph Isomorphism Network [37] and an actor-critic PPO algorithm where the GIN is shared between actor and critic networks. The method showed superior performance against

other traditional manually designed priority dispatching rule baselines, outperforming them by a large margin.

3.1 Conclusion

In conclusion, GRL is very promising field, where several different applications and techniques were already studied. GNNs architectures such as GCN have been extensively applied with DRL algorithms for enabling feature extracting from graph-based state representations [32, 41]. Architectures such as the GraphSAGE and other attention-based have also been successfully applied with very promising results [53, 33] in comparison with GCNs. However, less research regarding their integration with DRL algorithms was discovered. This suggests that a possible improvement and research direction in the development of GRL techniques might be connected with exploring the use of different GNNs architectures and using the rising attention-based techniques.

References

[1] Eugene Charniak. *Introduction to Deep Learning*. The MIT Press, Cambridge, Massachusetts London, England, 2018.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts, 2016.

[3] Feedforward Neural Network. https://orgs.mines.edu/daa/wp-content/uploads/sites/38/2019/08/1_Gh5PS4R_A5drl5ebd_gNrg@2x.jpg.

[4] Emma Brunskill. CS234: Reinforcement Learning Winter 2023.

[5] Richard S. Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts, nachdruck edition, 2014.

[6] Miguel Morales. *Grokking Deep Reinforcement Learning*. Manning, Shelter Island [New York], 2020.

[7] OpenAI. Spinning Up Documentation. https://spinningup.openai.com/en/latest/index.html.

[8] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, December 2017.

[9] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1928–1937. PMLR, June 2016.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[12] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, July 2019.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870. PMLR, July 2018.

[14] Zhiyuan Liu and Jie Zhou. *Introduction to Graph Neural Networks*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Springer International Publishing, Cham, 2020.

[15] Liang Zhao, Lingfei Wu, Peng Cui, and Jian Pei. Representation Learning. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 3–15. Springer Nature, Singapore, 2022.

[16] Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Nature Singapore, Singapore, 2022.

[17] William L Hamilton. Graph Representation Learning.

[18] Peng Cui, Lingfei Wu, Jian Pei, Liang Zhao, and Xiao Wang. Graph Representation Learning. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 17–26. Springer Nature, Singapore, 2022.

[19] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. Graph Neural Networks. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 27–37. Springer Nature, Singapore, 2022.

[20] Jian Tang and Renjie Liao. Graph Neural Networks for Node Classification. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 41–61. Springer Nature, Singapore, 2022.

[21] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, February 2017.

[22] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, February 2018.

[23] Hassan Farhangi. The path of the smart grid. *IEEE Power and Energy Magazine*, 8(1):18–28, January 2010.

[24] Tamilmaran Vijayapriya and Dwarkadas Pralhadas Kothari. Smart Grid: An Overview. *Smart Grid and Renewable Energy*, 02(04):305–311, 2011.

[25] R. Bayindir, I. Colak, G. Fulli, and K. Demirtas. Smart grid technologies and applications. *Renewable and Sustainable Energy Reviews*, 66:499–516, December 2016.

[26] Peng Li, Wenqi Huang, Zhen Dai, Jiaxuan Hou, Shang Cao, Jiayu Zhang, and Junbin Chen. A Novel Graph Reinforcement Learning Approach for Stochastic Dynamic Economic Dispatch under High Penetration of Renewable Energy. In *2022 4th Asia Energy and Electrical Engineering Symposium (AEEES)*, pages 498–503, March 2022.

[27] Xinpei Chen, Tao Yu, Zhenning Pan, Zihao Wang, and Shengchun Yang. Graph representation learning-based residential electricity behavior identification and energy management. *Protection and Control of Modern Power Systems*, 8(1):1–13, December 2023.

[28] Qiang Xing, Yan Xu, and Zhong Chen. A Bilevel Graph Reinforcement Learning Method for Electric Vehicle Fleet Charging Guidance. *IEEE Transactions on Smart Grid*, 14(4):3309–3312, July 2023.

[29] Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *Computer Communications*, 196:184–194, December 2022.

[30] Zhen Gao, Lei Yang, and Yu Dai. Fast Adaptive Task Offloading and Resource Allocation in Large-Scale MEC Systems via Multi-Agent Graph Reinforcement Learning. *IEEE Internet of Things Journal*, pages 1–1, 2023.

[31] Nan Li, Alexandros Iosifidis, and Qi Zhang. Graph Reinforcement Learning-based CNN Inference Offloading in Dynamic Edge Computing. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 982–987, December 2022.

[32] Junbin Chen, Tao Yu, Zhenning Pan, Mengyue Zhang, and Bairong Deng. A scalable graph reinforcement learning algorithm based stochastic dynamic dispatch of power system under high penetration of renewable energy. *International Journal of Electrical Power & Energy Systems*, 152:109212, October 2023.

[33] Qiang Xing, Zhong Chen, Tian Zhang, Xu Li, and KeHui Sun. Real-time optimal scheduling for active distribution networks: A graph reinforcement learning method. *International Journal of Electrical Power & Energy Systems*, 145:108637, February 2023.

[34] Daner Hu, Zichen Li, Zhenhui Ye, Yonggang Peng, Wei Xi, and Tiantian Cai. Multi-agent graph reinforcement learning for decentralized Volt-VAR control in power distribution systems. *International Journal of Electrical Power & Energy Systems*, 155:109531, January 2024.

[35] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow. Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[36] Hongyang Gao and Shuiwang Ji. Graph U-Nets. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2083–2092. PMLR, May 2019.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks?, February 2019.

[38] Mingshuo Nie, Dongming Chen, and Dongqi Wang. Reinforcement Learning on Graphs: A Survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(4):1065–1082, August 2023.

[39] Qiang Xing, Yan Xu, Zhong Chen, Ziqi Zhang, and Zhao Shi. A Graph Reinforcement Learning-Based Decision-Making Platform for Real-Time Charging Navigation of Urban Electric Vehicles. *IEEE Transactions on Industrial Informatics*, 19(3):3284–3295, March 2023.

[40] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combin-

ing Improvements in Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018.

[41] Fanfei Chen, John D. Martin, Yewei Huang, Jinkun Wang, and Brendan Englot. Autonomous Exploration Under Uncertainty via Deep Reinforcement Learning on Graphs. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6140–6147, October 2020.

[42] Lixiong Leng, Jingchen Li, Haobin Shi, and Yi'an Zhu. Graph convolutional network-based reinforcement learning for tasks offloading in multi-access edge computing. *Multimedia Tools and Applications*, 80(19):29163–29175, August 2021.

[43] Zhongxia Yan, Jingguo Ge, Yulei Wu, Liangxiong Li, and Tong Li. Automatic Virtual Network Embedding: A Deep Reinforcement Learning Approach With Graph Convolutional Networks. *IEEE Journal on Selected Areas in Communications*, 38(6):1040–1057, June 2020.

[44] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, July 2020.

[45] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. XGNN: Towards Model-Level Explanations of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 430–438, New York, NY, USA, August 2020. Association for Computing Machinery.

[46] Zhiqing Tang, Jiong Lou, Fuming Zhang, and Weijia Jia. Dependent Task Offloading for Multiple Jobs in Edge Computing. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, August 2020.

[47] Tianqiao Zhao and Jianhui Wang. Learning Sequential Distribution System Restoration via Graph-Reinforcement Learning. *IEEE Transactions on Power Systems*, 37(2):1601–1611, March 2022.

[48] Bangji Fan, Xinghua Liu, Gaoxi Xiao, Yu Kang, Dianhui Wang, and Peng Wang. Attention-Based Multi-Agent Graph Reinforcement Learning for Service Restoration. *IEEE Transactions on Artificial Intelligence*, pages 1–15, 2023.

[49] Yongdong Chen, Youbo Liu, Junbo Zhao, Gao Qiu, Hang Yin, and Zhengbo Li. Physical-assisted multi-agent graph reinforcement learning enabled fast voltage regulation for PV-rich active distribution network. *Applied Energy*, 351:121743, December 2023.

[50] Qiang Xing, Zhong Chen, Ziqi Zhang, Ruisheng Wang, and Tian Zhang. Modelling driving and charging behaviours of electric vehicles using a data-driven approach combined with behavioural economics theory. *Journal of Cleaner Production*, 324:129243, November 2021.

[51] Tao Qian, Chengcheng Shao, Xiuli Wang, and Mohammad Shahidehpour. Deep Reinforcement Learning for EV Charging Navigation by Coordinating Smart Grid and Intelligent Transportation System. *IEEE Transactions on Smart Grid*, 11(2):1714–1723, March 2020.

[52] Peidong Xu, Jun Zhang, Tianlu Gao, Siyuan Chen, Xiaohui Wang, Huaiguang Jiang, and Wenzhong Gao. Real-time fast charging station recommendation for electric vehi-

cles in coupled power-transportation networks: A graph reinforcement learning method. *International Journal of Electrical Power & Energy Systems*, 141:108030, October 2022.

[53] Yangzhou Pei, Jun Yang, Jundong Wang, Peidong Xu, Ting Zhou, and Fuzhang Wu. An emergency control strategy for undervoltage load shedding of power system: A graph deep reinforcement learning method. *IET Generation, Transmission & Distribution*, 17(9):2130–2141, May 2023.

[54] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 1621–1632. Curran Associates, Inc., 2020.