5-2024

Exploring Graph Neural Networks in Reinforcement Learning: A Comparative Study on Architectures for Locomotion Tasks

Gaukhar Nurbek
The University of Texas Rio Grande Valley

EXPLORING GRAPH NEURAL NETWORKS IN REINFORCEMENT LEARNING: A COMPARATIVE STUDY ON ARCHITECTURES FOR LOCOMOTION TASKS

A Thesis

by

GAUKHAR NURBEK

Submitted in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE

Major Subject: Computer Science

The University of Texas Rio Grande Valley

May 2024

EXPLORING GRAPH NEURAL NETWORKS IN REINFORCEMENT LEARNING: A COMPARATIVE STUDY ON ARCHITECTURES FOR LOCOMOTION TASKS

A Thesis
by
GAUKHAR NURBEK

COMMITTEE MEMBERS

Dr. Dongchul Kim
Chair of Committee

Dr. Emmet Tomai
Committee Member

Dr. Eric Enriquez
Committee Member

May 2024

ABSTRACT

Nurbek, Gaukhar, Exploring Graph Neural Networks in Reinforcement Learning: A Comparative Study on Architectures for Locomotion Tasks. Master of Science (MS), May, 2024, 29 pp., 2 tables, 9 figures, 54 references.

Deep Reinforcement learning (DRL) has gained importance in optimizing control policies, while Graph Neural Networks (GNNs) offer a robust approach for modeling complex relationships within systems represented as graphs. This thesis investigates the integration of DRL and GNNs to optimize control policies for robotic tasks, with a focus on locomotion. It compares static and dynamic GNN architectures for control policy predictions, revealing their strengths and limitations in adapting to locomotion predictions. The study assesses the impact of model structure complexity on GNNs' predictive capabilities, showcasing how intricate model structure can maximize GNNs' potential in capturing spatial and relational dependencies when learning control policies. Through comprehensive experiments and analysis, this research contributes insights into the optimal integration of GNNs in DRL, particularly in continuous locomotion tasks, addressing a gap in the literature. It also introduces a new implementation for conducting RL-GNN experiments with a 3-D simulation physics engine by employing PyTorch, PyTorch Geometric, and PyTorch Geometric Temporal, enhancing the methodological repertoire and promoting reproducibility in research. This study advances the understanding and application of DRL-GNN frameworks in robotics, facilitating the development of more intelligent and adaptive robotic systems.

DEDICATION

Dedicated to my family.

ACKNOWLEDGMENTS

I want to thank everyone who has supported me during my studies.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

Artificial intelligence (AI) stands at the forefront of technological innovation, reshaping the landscape of numerous industries and domains. At its core, AI seeks to enable machines to perform tasks that traditionally require human intelligence, ranging from simple pattern recognition to complex decision-making processes. Various AI subfields contribute to this mission. Natural language processing (NLP), for instance, focuses on enabling machines to understand and generate human language [1], facilitating applications such as machine translation[42], sentiment analysis [19], and chatbots, like ChatGPT [21]. Computer vision is another crucial area, where AI algorithms are trained to interpret and analyze visual data, enabling tasks like object recognition[38], image classification [17], and scene understanding [14]. The field of robotics integrates AI techniques to create intelligent, autonomous systems capable of interacting with and operating in the physical world [26]. Additionally, reinforcement learning, a subfield of machine learning, has emerged as a powerful paradigm within the realm of AI, drawing inspiration from behavioral psychology to teach agents to make sequential decisions by interacting with their environment. Unlike other machine learning techniques that rely heavily on labeled data, reinforcement learning enables autonomous agents to learn optimal behaviors through trial and error, guided by feedback in the form of rewards or penalties [37]. These subfields, working in harmony, are driving the advancements in AI, making it a significant tool in addressing complex problems and enhancing human capabilities across various domains.

In recent years, deep reinforcement learning (DRL) has gained significant attention due to its ability to handle high-dimensional and continuous state and action spaces [2]. DRL leverages deep neural networks to approximate complex functions that map states to actions, enabling agents

to learn intricate strategies and policies. The fusion of deep learning with reinforcement learning has led to remarkable breakthroughs in various domains, from mastering complex board games like Go and Atari to controlling autonomous vehicles and robotics [35], [24], [16]. The success of DRL can be attributed to its capacity to automatically learn hierarchical representations of raw sensory inputs, facilitating the extraction of meaningful features and patterns essential for decision-making.

In the realm of robotics, DRL has revolutionized the way robots learn to perform tasks in simulated environments and physical settings[41], [40]. Utilizing physics engines and realistic simulations, DRL allows robots to learn complex skills and behaviors without the need for extensive manual programming. From grasping objects to navigating dynamic environments, robots trained with DRL techniques demonstrate robustness and adaptability in various scenarios [18], [52].

Control policies play a pivotal role in DRL applications in robotics, serving as the backbone for decision-making and action execution by robotic agents. These policies dictate how robots interact with their environment, specifying the actions to be taken in response to observed states to maximize cumulative rewards. In the context of DRL, control policies are often represented by neural networks, which map environmental observations to corresponding actions. Through iterative exploration and exploitation, DRL algorithms train these policies to learn effective strategies for accomplishing tasks ranging from manipulation and locomotion to navigation and perception. The challenge lies in designing control policies that balance exploration and exploitation, ensuring robustness, adaptability, and efficiency in varying environmental conditions [34], [2].

The integration of graph neural networks (GNNs) with DRL has shown promise in enhancing the learning capabilities of agents, particularly in scenarios where the environment can be naturally represented as a graph. GNNs excel in capturing the structural dependencies and relational information inherent in graph-structured data, making them well-suited for tasks such as modeling complex relationships in social networks, biological systems, and physical environments [48]. In the context of robotics, where spatial relationships and connectivity play a crucial role, GNNs offer a powerful framework for learning representations that can effectively capture the underlying structure of the environment. By leveraging GNNs in addition to DRL, robots can acquire robust

and adaptive policies that exploit the rich spatial and relational information available in their surroundings. One of the pioneers in utilizing GNNs in DRL for locomotion is [44], where the agent is provided structural information from 3-D model simulation in physics engine [40].

This thesis contributes to the field of robotics by investigating the synergies between DRL and GNNs to enhance control policies for diverse robotic tasks, including locomotion such as walking. Firstly it compares static and temporal GNN architectures for control policy predictions, shedding light on their respective strengths and weaknesses in adapting to locomotion predictions. After that, it explores the potential of integrating DRL and GNNs, leveraging the structural and relational information encoded within GNNs to optimize control policies. It delves into the impact of agent structure complexity on the predictive capabilities of GNNs, demonstrating how complex agent architectures can maximize the potential of GNNs in capturing and utilizing spatial and relational dependencies. Moreover, this study provides a new implementation to DRL-GNN experiments, which utilizes PyTorch, PyTorch Geometric, PyTorch Geometric Temporal libraries, previously not used for similar robotic DRL-GNN experiments. Through these contributions, this research aims to advance the understanding and application of DRL-GNN frameworks in robotics, paving the way for more intelligent and adaptive robotic systems.

CHAPTER II

LITERATURE REVIEW

2.1 Graph Neural Networks

Early studies on Graph Neural Networks (GNNs) were motivated by [36], where neural networks were applied to directed graphs for the first time. The [9] initially outlined the notion of graph neural networks. This notion was further used and elaborated for the idea of the GNN model that was proposed by authors in [33]. The proposed GNN model extends existing neural network methods to operate on data represented in graph domains. They derived a supervised learning algorithm that estimates the parameters of the GNN model.

The breakthrough came with the introduction of Graph Convolutional Networks (GCNs) by Thomas Kipf and Max Welling [15]. GCNs utilized a localized first-order approximation of spectral graph convolutions, enabling information propagation across neighboring nodes. This allowed neural networks to be applied to graphs in a scalable manner. GCNs are the foundation for later GNN architectures.

Researchers were able to refine the concept of information aggregation, allowing nodes to selectively attend to their neighbors. Authors in [10] introduced a GraphSAGE, framework for inductive learning on large graphs, where node embeddings are learned considering neighboring sampling strategies. While authors of [43] proposed a novel attention mechanism that enables nodes to weigh neighbors differently during message aggregation, improving the expressive power of GNNs. Graph Isomorphism Networks (GINs) were introduced in [49] where they address the graph isomorphism problem, providing insights into learning structural representations of graphs that are invariant to their topological differences.

GNNs found applications in various fields, including social network analysis[5, 6], recommendation systems[51, 20], drug discovery[53, 11], reinforcement learning[44, 22] and other areas. Researchers continue to develop GNN architectures tailored for specific tasks, leading to the creation of different versions of GNN and improvements.

2.2 Reinforcement Learning

[45] is one of the foundational works in Reinforcement Learning (RL), introducing the Q-learning algorithm for model-free reinforcement learning. Q-learning is a way for agents to learn how to optimally act in controlled Markovian domains. TD-Gammon introduced in [39] demonstrated the application of temporal difference learning, a core concept in RL, in training neural networks to play backgammon at a high level.

In [24] Deep Q-Networks (DQN) algorithm was introduced, where Q-learning was combined with deep neural networks, achieving human-level performance on multiple Atari 2600 games. Authors of [25] demonstrated how Deep Reinforcement Learning (DRL) algorithms can learn directly from high-dimensional sensory inputs, demonstrating the possibility of usage of RL in complex environments.

Further work in RL was introduced in Policy Gradient Methods and Actor-Critic Architectures. [23] introduced asynchronous advantage actor-critic (A3C), an algorithm that allows agents to learn from multiple parallel instances of the environment, significantly speeding up training. [34] presented the Proximal Policy Optimization (PPO) algorithm, a policy optimization technique that demonstrated stable and efficient learning in complex environments.

There are works on real-world and robotics applications, such as [27] demonstrated RL algorithms applied to robotics, showcasing how DRL methods allow robots to learn complex manipulation tasks.

In recent years, RL has continued to advance, with research focusing on improving sample efficiency, stability, and safety of algorithms. Applications of reinforcement learning have expanded into various fields, including robotics, autonomous vehicles, finance, healthcare, and more.

2.3 Graph Neural Networks in Reinforcement Learning

There have been few works developed in applying Graph Neural Networks (GNNs) to Reinforcement Learning (RL) continuous control tasks. One of them focuses on applying GNNs in RL policy is NerveNet [44]. It addresses the problem of learning structured policies for continuous control. Author of [44] proposes a model NerveNet that utilizes the structure of an agent. The NerveNet model serves as an agent's policy network that propagates information over the structure of the agent and then predicts the action. The model was compared with the state-of-the-art models in Mujoco environments. They proposed two types of environments to benchmark two types of transfer learning which are: 1) size and disability transfer, and 2) multi-tasking learning.

Another method was suggested by the authors of [3]. They propose a method for high-dimensional continuous control that freezes parameters in selected parts of the network that are based on NerveNet [44]. The paper claims to find a solution to the network's overfitting problem.

Authors of the most recent paper [22] propose a new approach called Feudal Graph RL, where they adopt Feudal RL that develops control actions based on a hierarchical message-passing process. In the architecture, they propose high-level decisions at the top level of the hierarchy that are propagated through a layered graph representing the hierarchy of policies. The authors formalized the proposed framework and provided Proof Of Concept experiments on MuJoCo [40] locomotion tasks.

While these studies have provided valuable insights into applications of GNNs in RL in locomotion tasks, there remains a gap in the literature regarding a comprehensive comparative analysis of different GNN architectures specifically tailored for locomotion tasks in RL.

CHAPTER III

BACKGROUND

This section introduces notations and definitions used in Deep Learning, various types of GNNs, and RL to provide a background.

3.1 Multi-Layer Perceptron

Feedforward neural networks (FNNs) or multi-layer perceptrons (MLPs) aim to approximate a function f^*, where, for example, for a classifier, $y = f^*(x)$ maps x to a category y. MLP defines a mapping $y = f(x, \theta)$ where θ is a learnable parameter that provides an optimal approximation of the function f.

FNNs are called ”networks” because they are constructed by combining various functions in a directed acyclic graph. These functions, represented by $f^{(1)}, f^{(2)}, f^{(3)}$, and so on, are connected in a chain-like structure, denoted as $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. In this structure, $f^{(1)}$ is the first layer, $f^{(2)}$ is the second layer, and so forth. The depth of the model is determined by the overall length of this chain, and this terminology led to the term ”deep learning.”

The final layer in a FNN is called the ”output layer.” During training, the network is trained to approximate a function $f^*(x)$. The training data provides noisy, approximate examples of $f^*(x)$ at different points x, accompanied by labels $y = f^*(x)$. The training examples specify what the output layer should do at each point x by producing a value close to y. However, the behavior of the other layers is not directly specified by the training data. These unspecified layers are known as ”hidden layers.” The learning algorithm must determine how to use these hidden layers effectively to implement an approximation of f^* since the training data do not provide specific instructions for each layer. [8]

3.2 Reinforcement Learning

Reinforcement learning involves acquiring the knowledge of determining appropriate actions, to maximize a numerical reward signal. The problem in reinforcement learning is formalized using concepts from dynamical system theories, such as optimal control of partially known Markov decision processes [37]. A Markov Decision Process (MDP) is used as a mathematical framework in reinforcement learning to represent decision-making scenarios occurring in stochastic environments. It offers a structured approach to define and explore sequential decision-making processes where an agent engages with its surroundings. In MDP there exists an agent that interacts with the environment. They interact at a discrete time step, $t = 0, 1, 2, 3, ...$ At each time step t agent receives state $S_t \in S$, which is the environment's representation, and based on the state agent selects an action, $A_t \in A(s)$. In the next time step after action was taken $t+1$ agent receives a reward, $R_{t+1} \in R$, and transitions into a new state, S_{t+1}. The sequence of MDP looks like below [37]:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, R_3, ... \tag{3.1}$$

Policy π is a function that maps states to a probability distribution over actions: $\pi : S \rightarrow p(A = a|S)$. Finding optimal policy π^*, that provides maximum reward for all states is a goal of reinforcement learning [2].

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E[R|\pi] \tag{3.2}$$

3.3 Proximal Policy Optimization

Proximal policy optimization (PPO) is introduced in [34]. Unlike traditional policy gradient methods, the PPO iteratively interacts with the environment to collect data and then optimizes an objective function through stochastic gradient ascent. While standard policy gradient methods perform a single gradient update for each data sample, the suggested method applies a distinctive objective function that allows for multiple epochs of mini-batch updates.

A suggested objective function is as follows:

$$L_t^{cLIP+VF+S}(\theta) = \widehat{E}_t[L_t^{cLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{3.3}$$

where c_1, c_2 are coefficients, S is an entropy bonus, $L_t^{VF} = (V_\theta(s_t) - V_t^{targ})^2$ is a squared-error loss. In order to run the policy for T timesteps, where T is less than the episode length, and use the collected samples for an update, an advantage estimator needs to be calculated that does not look beyond timestep T. PPO algorithm uses the following estimator:

$$\widehat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{3.4}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ and r denotes the probability ratio and $V(s)$ denotes state value function.

3.4 Graph Neural Network

Graph neural networks (GNNs) are neural networks operating on graph-structured data. Many examples in the real world can be represented as graph-structured data. For instance, in social networks users can be represented as nodes, and connections between users can be represented as edges. These nodes can have multiple features, it could be a user's age, number of followers, number of posts, number of likes, etc. Given connections between nodes and their features GNNs can utilize them to make predictions.

A graph G consists of a non-empty finite set of vertices or nodes and a finite set of edges, that are pairs of connected nodes [46]. $G = (V, E)$, where V is a set of nodes and E is the set of edges between nodes. Each node can be denoted as $v_i \in V$ and edge between v_i, v_j is denoted as $e_i = (v_i, v_j) \in E$. The adjacency matrix A is a $n \times n$ matrix, where $A_{ij} = 1$ if $e_{ij} \in E$ else $A_{ij} = 0$ [48]. There are directed and undirected graphs. In GNNs, graphs are treated as directed, and even in the case of undirected graphs, they are typically converted into directed graphs by considering edges that connect nodes in both directions.

The goal of GNNs is to iteratively update the node representations by aggregating the representations of node neighbors and their representations in the previous iteration.[47] This can be done using message passing. Message passing refers to the process by which information between nodes is exchanged. This process consists of three steps [32]:

- For each node $v \in G$ collect messages as neighboring node embedding.
- Aggregate collected messages.
- Pass aggregated messages through an update function, which is a learned neural network.

There are 3 levels of prediction tasks performed using GNNs. First is graph level prediction, second is node level prediction and third is edge level prediction, where all of the levels include both regression and classification tasks.

3.5 Graph Convolutional Network

The authors in [15] propose an efficient semi-supervised learning method for graph-structured data, utilizing a variant of convolutional neural networks designed to work directly on graphs. Their approach is motivated by a localized first-order approximation of spectral graph convolutions. This model scales linearly with the number of graph edges and captures both local graph structures and node features in its hidden layer representations.

Authors of [15] introduced multi-layer Graph Convolutional Network (GCN) with the following layer-wise propagation rule:

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

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(x)$ denotes an activation function, for example the $\mathrm{ReLU}(x) = \max(0,x)$. $H^{(l)} \in R^{ND}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$.

Assuming $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, example of 2-layer GCN architecture used for semi-supervised node classification is shown below:

$$Z = f(X,A) = \text{softmax}(\hat{A}\ \text{ReLU}(\hat{A}XW^{(0)})W^{(1)})$$

3.6 Graph Attention Networks

The authors of [43] introduce Graph Attention Networks (GATs). GATs utilize masked self-attentional layers to overcome limitations in prior methods based on graph convolutions or their approximations. By enabling nodes to attend over their neighborhoods' features and specify different weights to different nodes without costly matrix operations or prior knowledge of the graph structure, GATs simultaneously address multiple challenges in spectral-based graph neural networks. The model applies to both inductive and transductive problems.

The graph attention layer is the sole layer in GAT architectures. The input to the layer is a set of node features, $h = \{\vec{h_1}, \vec{h_2}, ..., \vec{h_N}\}, \vec{h_i} \in R^F$, where N is the number of nodes, and F is the number of features per node. The output of the layer is the new set of node features $h' = \{\vec{h'_1}, \vec{h'_2}, ..., \vec{h'_N}\}, \vec{h'_i} \in R^{F'}$.

To obtain expressive power we need to apply weight matrix $W \in R^{F' \times F}$ for every node and then perform self-attention on each node a that computes attention coefficients $e_{ij} = a(W\vec{h_i}, W\vec{h_j})$ which shows the importance of node j's features to node i. Using masked attention allows to utilization of structural information, which means computing e_{ij} only for nodes $j \in N_i$, where N_i is the neighborhood of the node i in the graph. After that normalization is applied to all choices of j, to make coefficients easily comparable. $\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i}(e_{ik})}$ Attention mechanism a used in a paper is a single-layer FNN or MLP, with a weight vector $\vec{a} \in R^{2F'}$ and LeakyReLU activation function.

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\vec{a}^T[W\vec{h_i}||W\vec{h_j}]))}{\sum_{k \in N_i}\exp(\text{LeakyReLU}(\vec{a}^T[W\vec{h_i}||W\vec{h_k}]))}$$

where T is a transposition and || is the concatenation.

Once α_{ij} is computed the results are used to get the output features of nodes, which is a linear combination of features $\vec{h}'_i = \sigma(\sum_{j \in N_i} \alpha_{ij} W \vec{h}_j)$. For the multi-head attention mechanism depending on the number of attention heads K, the output features will be equal to the concatenation of all features for K heads. For the last output layer average along K heads will be taken.

3.7 Spatio-Temporal Graph Neural Networks

Spatio-temporal graph neural networks (STGNNs) are extensions of GNNs that learn temporal representations of the graph structure to take the spatial and temporal dimensions of data into consideration. [31]

There are static and dynamic spatio-temporal graphs [12]. In static spatio-temporal graphs, only features of nodes change, but connections stay the same, while in dynamic spatio-temporal graphs edges as well as node features can change.

The basic framework of STGNNs for predictive learning contains 3 main parts: Data Processing Module, Spatio Temporal Graph Learning Module (STGLM), and Task Aware Prediction Module (TPM). STGLM serves as the main component of STGNNs that combines spatial learning networks and temporal learning networks using the spatial-temporal fusion method. Spatial learning networks might be GCNs, GATs, etc. While the temporal learning networks may use RNNs, temporal convolutional networks (TCNs), or temporal self-attention networks (TSANs). [12]

3.8 Temporal Graph Networks

One of the STGNN algorithms to deal with dynamic graphs is Temporal Graph Networks (TGN) introduced in [29]. TGNs were developed for deep learning on dynamic graphs, where graphs are represented as a sequence of time events. TGNs utilize memory modules and graph-based operators which makes them computationally efficient.

A neural model for dynamic graphs can be regarded as an encoder-decoder pair. The encoder is a function that maps dynamic graphs to node embedding, while the decoder uses node embedding as input to make a task-specific prediction, for example, node classification [13].

A TGN encoder is applied on a continuous-time dynamic graph represented as a sequence

of time-stamped events and it produces the embedding for each time t, the embedding of the graph nodes $Z(t) = (z_1(t), ..., z_{n(t)}(t))$.

3.9 Temporal Graph Convolution Networks

The Temporal Graph Convolution Network (TGCN) was introduced in [54] for traffic forecasting since it has both spatial and temporal dependencies. In TGCN model first historical n time series data is used as input, while GCN is used to capture topological structure. After obtaining time series and spatial features, those are used as input to GRU model[4] where dynamic change is obtained by information transmission between the units that capture temporal feature, after which results are calculated using a fully connected layer.

CHAPTER IV

METHODOLOGY

This chapter describes methods and tools that were created to perform experiments and collect data. Three main components had to be developed to achieve this. Firstly, a custom reinforcement learning environment had to be implemented to simulate locomotion walking tasks and return observations in a structured way and it is described in section 4.1. Secondly, the proximal policy optimization (PPO) algorithm had to be implemented and the process is described in the section 4.2. Additionally, various types of policy network architectures, such as multi-layer perception, and multiple graph neural networks were developed, the details of which are provided in section 4.3. Details of the information flow connecting three components can be seen in Fig.4.1.

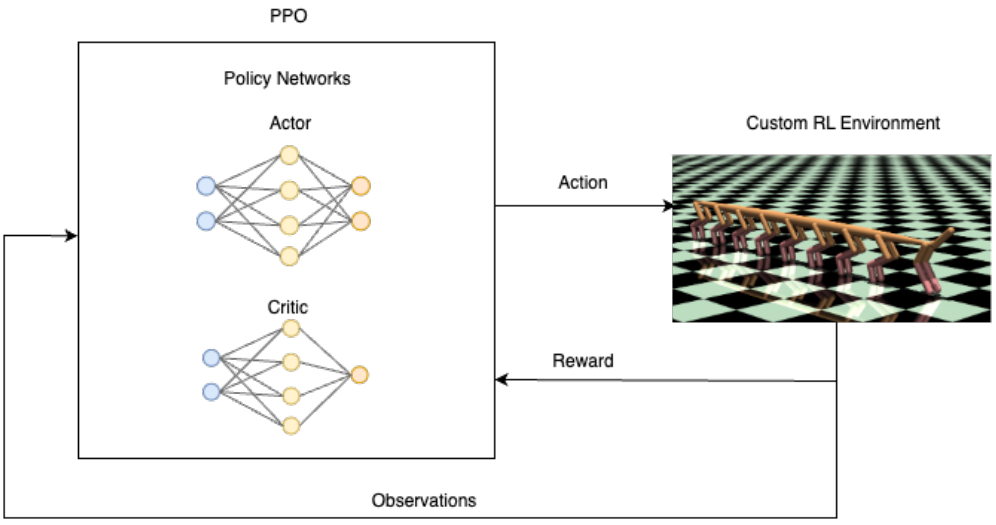

Figure 4.1: The figure shows the flow of information between the custom reinforcement learning environment, the proximal policy optimization, and the policy network.

4.1 Custom Reinforcement Learning Environments

A custom reinforcement learning (RL) environment was implemented using open source physics engine MuJoCo[40] and RL API Gymnasium[41]. There are two parts of the custom environment that was implemented. First is creating a customized 3-D model simulation to run in MuJoCo. Second part is defining actions, observations and reward of environment using Gymnasium and setting up exchange of information between MuJoCo model simulation and RL environment. For developed models, velocities and positions of joints of the models are passed as observations, and actions represent torque values applied at each joint.

Two types of custom environments were developed for this project. The first type feeds observations as concatenation from all joints. While, second type is constructed to model joint connections and returns structured observations in a matrix representing each joint's velocities and positions, as well as an array of edges between connected joints. This type of environment receives predictions from a GNN-based policy. The reward in both environments depends on the model moving forward at high speed. One of the implemented 18-model

4.2 PPO Implementation

The Proximal Policy Optimization (PPO) algorithm was chosen as the primary reinforcement learning technique for this thesis due to its effectiveness in policy optimization tasks. The main body of PPO implementation was referenced from [50]. However, the implementation needed customization to accommodate the characteristics of graph-structured observations. Graphs inherently possess complex relational information that traditional reinforcement learning algorithm implementations are not equipped to handle. To address this challenge, changes were made to enable graph-batch training, where multiple graphs are merged into one large graph and then passed into graph neural network policies.

4.3 Policy Network Implementation

In the PPO algorithm neural network is used as an actor and critic. The actor policy network takes observations as input and predicts the mean values of actions of the agent. The critic policy

network takes observations as an input and outputs the value of the function, which decides how good the actions were predicted. In this project multiple network architectures were implemented using PyTorch [28] for feed-forward neural network (FNN) and PyTorch Geometric [7] for static graph convolution network (GCN), PyTorch Geometric Temporal [30] for dynamic recurrent graph convolution network (RecGCN). FNN inputs are provided as concatenated observations from each of the joints of the model, in the case of GCN and RecGCN observations were provided in a 2-dimensional array, along with edges, that lists connected joints as a separate matrix. In Fig.4.2 layers of the RecGCN are shown.

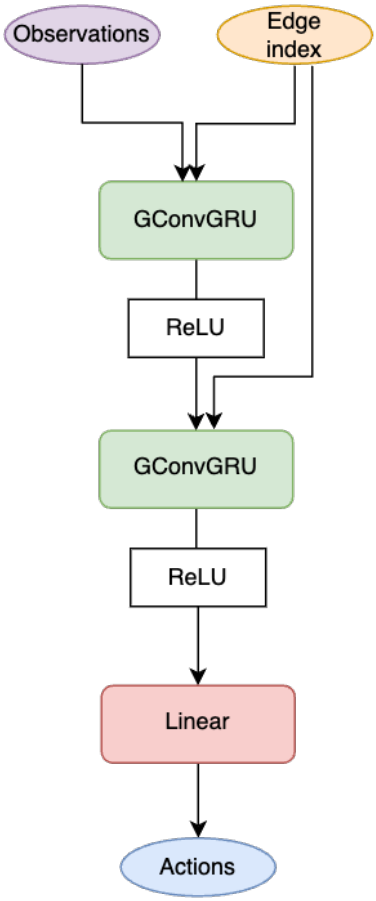

Figure 4.2: The figure shows general architecture of dynamic RecGCN used as PPO actor and critic networks

CHAPTER V

RESULTS

Experiment results are presented in this chapter. Experiments were done on a computer with an Intel Core i9-11900 processor running at 2.5 GHz using 32 GB RAM, running Linux Ubuntu 20.01. Training parameters used for the PPO algorithm are included in Tab. 5.1. Overall 3 parts of the results were produced. The first part explores different graph neural network (GNN) architectures most suitable for predicting control policies in a proximal policy optimization(PPO) algorithm for the task of walking given graph-structured observations. The second and third parts explore how the complex structure of the model affects the training of the graph-based and non-graph-based control policies.

Table 5.1: Training parameters used in PPO algorithm

Parameters	Value
Value discount factor γ	0.99
Starting learning rate	3e-4
PPO Clip value	0.2
GAE λ	0.95
Target KL	0.01
Timesteps per batch	2048
Max timesteps per episode	200
Total timesteps	10e6

5.1 Exploring GNN Architectures

There were explored static graph convolution network (GCN) and dynamic recurrent graph convolution network (RecGCN) networks as control policies in PPO. The hyperparameters used for this part of the experiments are shown in Tab. 5.2. Fig. 5.1 shows the reward curve obtained

from training a 6-legged creature for the task of walking forward. The training curve for GCN and RecGCN shows that dynamic RecGCN is better at predicting action mean values for locomotion control policy for custom 6-legged creatures.

Table 5.2: Hyperparameter settings for multi-legged creature training

FNN, GCN, RecGCN	Value
Network Shape	[64, 64]
Number of iterations per update	10
Number of hidden nodes	64
RecGCN	**Value**
K Chebyshev filter size	3

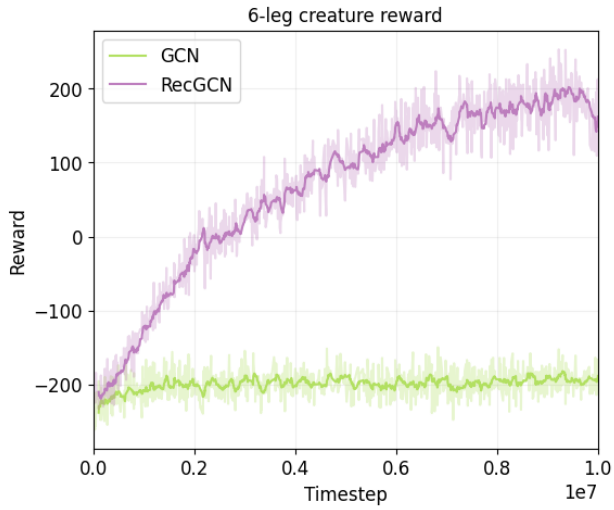

Figure 5.1: The figure shows a comparison between reward curves that were obtained by training static (GCN) and dynamic (RecGCN) GNNs for the 6-legged creature model. It appears that the dynamic GNN is better at predicting action mean values compared to the static GCN model.

5.2 Multi-legged MuJoCo Models

After discovering working GNN architecture, next step was to explore if complicating the structure of the model by adding additional legs to the creature, will allow us to fully utilize GNN's power and compare it with default FNN network used in PPO training. For this purpose, multiple MuJoCo models were created and trained to walk forward. 6, 10, and 18-legged creatures were made. 18 leg-creature is shown in Fig.5.2.

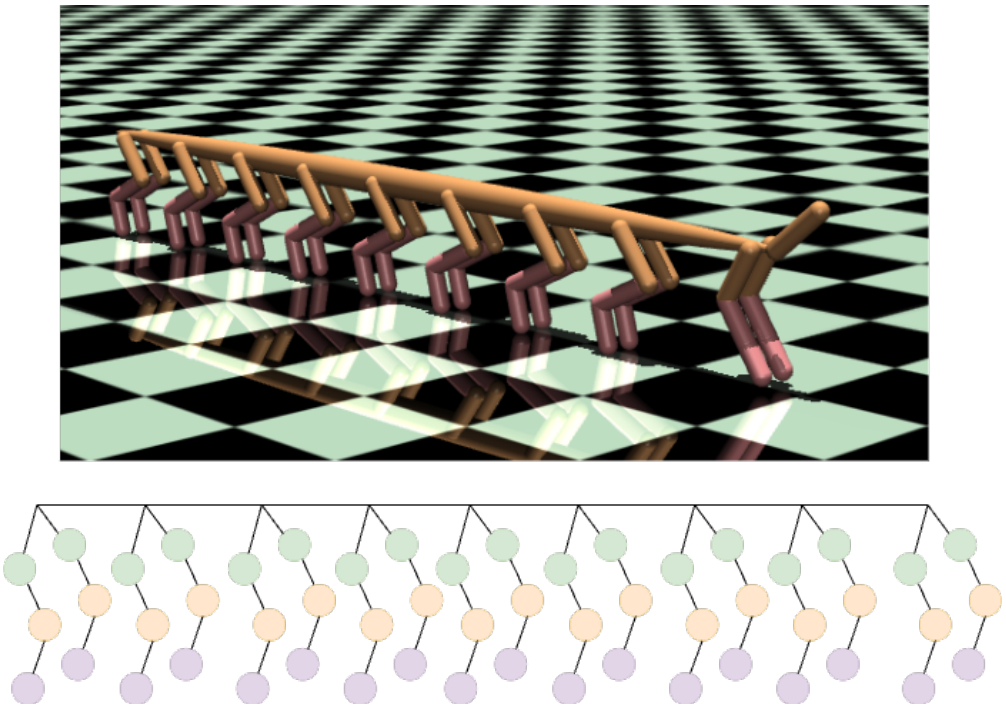

Figure 5.2: The figure shows an 18-legged creature model made in MuJoCo and its graph structure used for training GNN policy.

Hyperparameters used in training FNN and RecGCN as policies are shown in 5.2. However, in the case of training an 18-legged creature, K Chebyshev filter size was changed to 6. The training curve for 6, 10, and 18-legged models can be seen in Fig. 5.3, 5.4, 5.5.

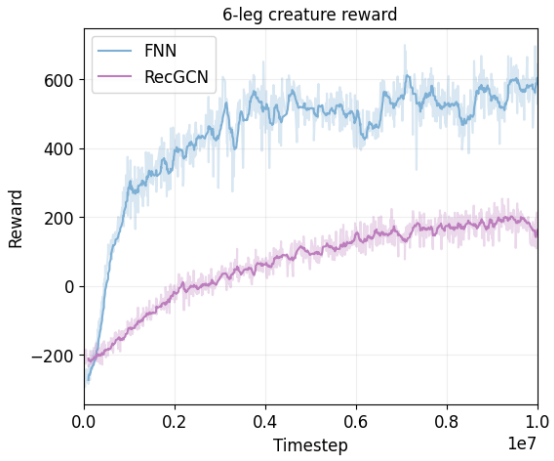

Figure 5.3: The figure shows reward curves trained using FNN and RecGCN for a 6-legged creature.

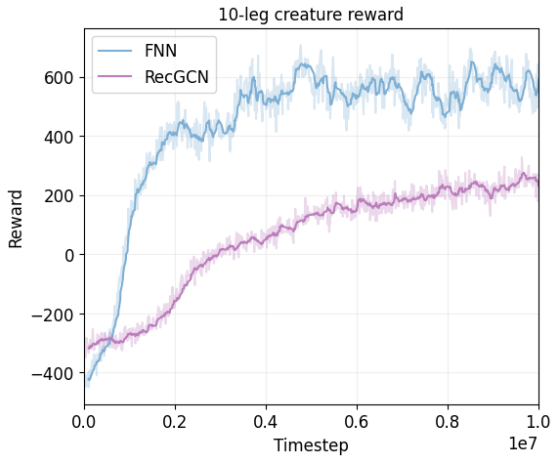

Figure 5.4: The figure shows reward curves trained using FNN and RecGCN for a 10-legged creature.

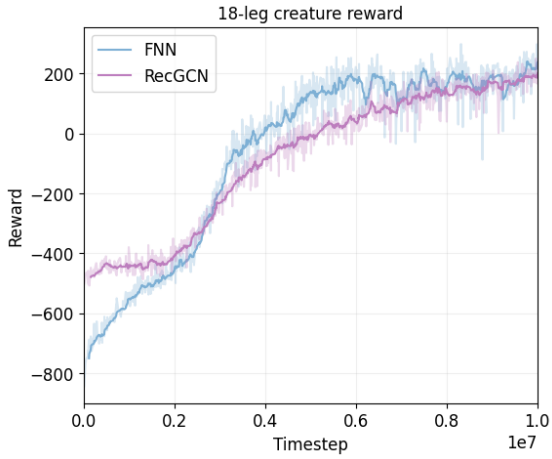

Figure 5.5: The figure shows reward curves trained using FNN and RecGCN for an 18-legged creature.

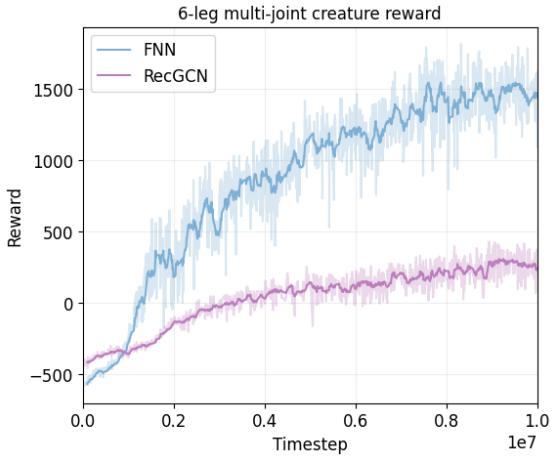

Figure 5.6: The figure shows reward curves trained using FNN and RecGCN for a 6-legged multi-joint creature.

5.3 Multi-Joint MuJoCo Models

In the third part of the results there were trained multi-joint models. Where instead of increasing the number of legs, the number of joints was increased to complicate the structure of the model. The figure of the model can be seen in Fig.5.7. Training hyperparameters were the same as in Section 5.2. Reward curves for training 6-legged multi-joint creature policies are shown in Fig. 5.6.

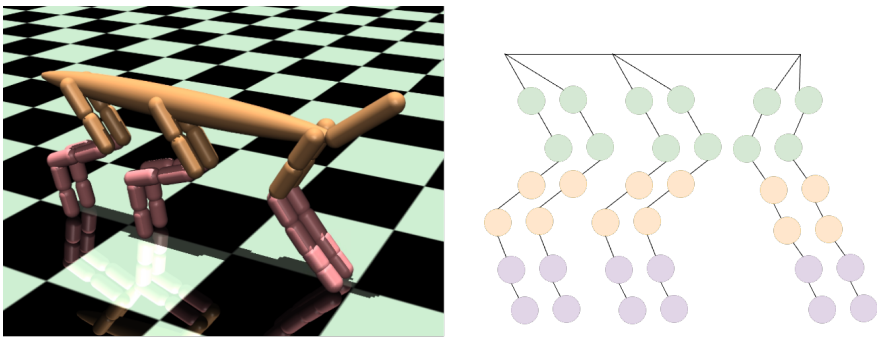

Figure 5.7: The figure shows an 18-legged creature model made in MuJoCo and its graph structure used for training GNN policy.

CHAPTER VI

DISCUSSION

This chapter delves into the interpretation and implications of the experimental results obtained in this study. The performance of various network policies in predicting actions is evaluated, focusing on the comparison between recurrent graph convolution networks (RecGCN), static graph convolution networks (GCN) and feedforward neural network (FNN).

From training different types of graph neural networks (GNN) as policy networks for a 6-legged creature walking forward task Fig.5.1 shows that the recurrent graph convolution networks (RecGCN) outperforms the static graph convolution network (GCN). This implies the effectiveness of RecGCN as a dynamic GNN model, as it recognizes the dynamic evolution of node values within a graph over time. Unlike static GNNs, which presume node values to be constant and regard any alteration as constituting a new, disconnected graph state, RecGCN's capability to adjust to fluctuating node values results in enhanced predictions and ultimately yields superior rewards.

In comparing the performance of policy networks across multiple models with 6, 10, and 18 legs clear patterns stand out. One of them is that the performance of FNN for 6 and 10-legged creatures appears superior to that of RecGCN as seen from Fig.5.3,5.4. However, upon increasing the number of legs to 18, a shift occurs where the performance of FNN notably declines, whereas RecGCN maintains a consistent level of performance. This observation suggests that RecGCN, which inherently considers structural information within the graph, possesses the capability to preserve performance even with increased complexity. The contrast in performance dynamics between FNN and RecGCN underscores the significance of leveraging structural information in enhancing the adaptability and robustness of policy networks across varying complexities of creature models.

For the multi-joint model, it appears that the FNN has effectively leveraged the capabilities of RecGCN as seen in 5.6. However, future studies may involve investigating the potential benefits of incorporating additional legs and additional joints into the model. By expanding the complexity of the model, there is a prospect for GNN-based policy networks to enhance their predictive accuracy in controlling policies. This extension could offer valuable insights into the scalability and adaptability of GNN architectures in addressing increasingly intricate control tasks. Nevertheless, this area warrants deeper investigation and is thus left for future studies to delve into comprehensively.

CHAPTER VII

CONCLUSION

This study introduces insights through a comparative analysis between static and temporal graph neural networks (GNNs) in reinforcement learning (RL) applications, specifically for robotic locomotion tasks. It has been found that temporal GNNs outperform static ones in predicting control policy values for locomotion tasks. Additionally, this study explores the impact of the structural complexity of the model on the training of both structured and non-structured policy networks in the Proximal Policy Optimization (PPO) algorithm, finding a connection between the model's structure and training effectiveness. These findings significantly contribute to the existing literature by deepening our understanding of the synergy between RL and GNNs in robotic locomotion tasks, paving the way for further exploration and development in this field.

In addition to the findings presented in this study, an important contribution lies in the novel implementation of the methodology. Utilizing PyTorch, PyTorch Geometric, and PyTorch Geometric Temporal for conducting RL-GNN experiments represents a significant advancement in the field. Notably, these software tools have not been previously utilized in similar experiments, underscoring the innovative nature of the approach used for this study. By leveraging these frameworks, not only was the methodological repertoire available to researchers expanded, but also the rigor and reproducibility of research was enhanced. This methodological innovation represents a valuable contribution to the ongoing discourse surrounding the synergy between reinforcement learning and graph neural networks.

REFERENCES

[1] James F. Allen. *Natural language processing*, page 1218–1222. John Wiley and Sons Ltd., GBR, 2003.

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *IEEE SIGNAL PROCESSING MAGAZINE, SPECIAL ISSUE ON DEEP LEARNING FOR IMAGE UNDERSTANDING (ARXIV EXTENDED VERSION)*, 2017.

[3] Charlie Blake, Vitaly Kurin, Maximilian Igl, and Shimon Whiteson. Snowflake: Scaling gnns to high-dimensional continuous control via parameter freezing. *CoRR*, abs/2103.01009, 2021.

[4] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi, editors, *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[5] Alex Davies and Nirav Ajmeri. Realistic synthetic social networks with graph neural networks, 2022.

[6] Leyan deng, Chenwang Wu, Defu Lian, Yongji Wu, and Enhong Chen. Markov-driven graph convolutional networks for social spammer detection. *IEEE Transactions on Knowledge and Data Engineering*, PP:1–1, 01 2022.

[7] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[11] Kehang Han, Balaji Lakshminarayanan, and Jeremiah Z. Liu. Reliable graph neural networks for drug discovery under distributional shift. *CoRR*, abs/2111.12951, 2021.

[12] Guangyin Jin, Yuxuan Liang, Yuchen Fang, Jincai Huang, Junbo Zhang, and Yu Zheng. Spatio-temporal graph neural networks for predictive learning in urban computing: A survey, 2023.

[13] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[14] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *CoRR*, abs/1511.02680, 2015.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[16] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017.

[18] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps, 2014.

[19] Xiaoxin Lin. Sentiment analysis of e-commerce customer reviews based on natural language processing. In *Proceedings of the 2020 2nd International Conference on Big Data and Artificial Intelligence*, ISBDAI '20, page 32–36, New York, NY, USA, 2021. Association for Computing Machinery.

[20] Han Liu, Yinwei Wei, Jianhua Yin, and Liqiang Nie. Hs-gcn: Hamming spatial graph convolutional networks for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 35(6):5977–5990, 2023.

[21] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Lin Zhao, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 1(2):100017, September 2023.

[22] Tommaso Marzi, Arshjot Khehra, Andrea Cini, and Cesare Alippi. Feudal graph reinforcement learning, 2023.

[23] 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 Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas 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*, 518:529–33, 02 2015.

[26] Luiz F. P. Oliveira, António P. Moreira, and Manuel F. Silva. Advances in agriculture robotics: A state-of-the-art review and challenges ahead. *Robotics*, 10(2), 2021.

[27] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation, 2019.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[29] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs, 2020.

[30] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models, 2021.

[31] Zahraa Al Sahili and Mariette Awad. Spatio-temporal graph neural networks: A survey, 2023.

[32] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[33] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[35] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[36] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[37] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction Second Edition*. The MIT Press, 2020.

[38] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[39] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, mar 1995.

[40] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control.

[41] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. gymnasium, March 2023.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[43] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[44] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[45] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[46] Robin J. Wilson. *Introduction to Graph Theory Fourth edition*. Longman, 1996.

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

[48] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021.

[49] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[50] Eric Yang Yu. Ppo-for-beginners, 2020. `https://github.com/ericyangyu/PPO-for-Beginners`.

[51] Wenhui Yu, Zixin Zhang, and Zheng Qin. Low-pass graph convolutional network for recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):8954–8961, Jun. 2022.

[52] Junjie Zeng, Rusheng Ju, Long Qin, Yue Hu, Quanjun Yin, and Cong Hu. Navigation in unknown dynamic environments based on deep reinforcement learning. *Sensors*, 19(18), 2019.

[53] Shuo Zhang, Yang Liu, and Lei Xie. Molecular mechanics-driven graph neural network with multiplex graph for molecular structures, 2020.

[54] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, September 2020.

VITA

Gaukhar Nurbek received a Bachelor of Engineering and Technology with a major in Information Systems from Kazakh-British Technical University in May 2018. Gaukhar earned a Master of Science in Interdisciplinary Studies with a concentration in Science and Technology at The University of Texas Rio Grande Valley (UTRGV) in May 2021, where she was awarded a Presidential Graduate Research Assistantship. Gaukhar's research has been in close collaboration with the Laser Interferometer Gravitational-Wave Observatory (LIGO) Scientific Collaboration's Supernova working group and it resulted in two papers' publication. In the next year, she worked as a research assistant at UTRGV. Gaukhar earned her Master of Science in Computer Science in May 2024. During her studies, she did a 12-week internship as a software engineer at Uber. Gaukhar is planning to pursue her research interests and pursue a Ph.D. degree in Computer Science. Her email is gaukharanurbek@gmail.com.