UNIVERSITY OF SALERNO

DEPARTMENT OF INFORMATION ENGINEERING ED
ELECTRICAL AND APPLIED MATHEMATICS

MASTER'S DEGREE COURSE IN COMPUTER ENGINEERING
(LM-32)

MACHINE LEARNING

Training a robotic arm to play Ping Pong:

"Project Work"

TEAM 22

NOME & COGNOME	E-MAIL
Antony Storti	a.storti2@studenti.unisa.it
Paola Saggiomo	p.saggiomo1@studenti.unisa.it
Miriam Valentino	m.valentino21@studenti.unisa.it
Damiano Noschese	d.noschese3@studenti.unisa.it

Academic Year 2023 – 2024

Summary

CHAPTER 1

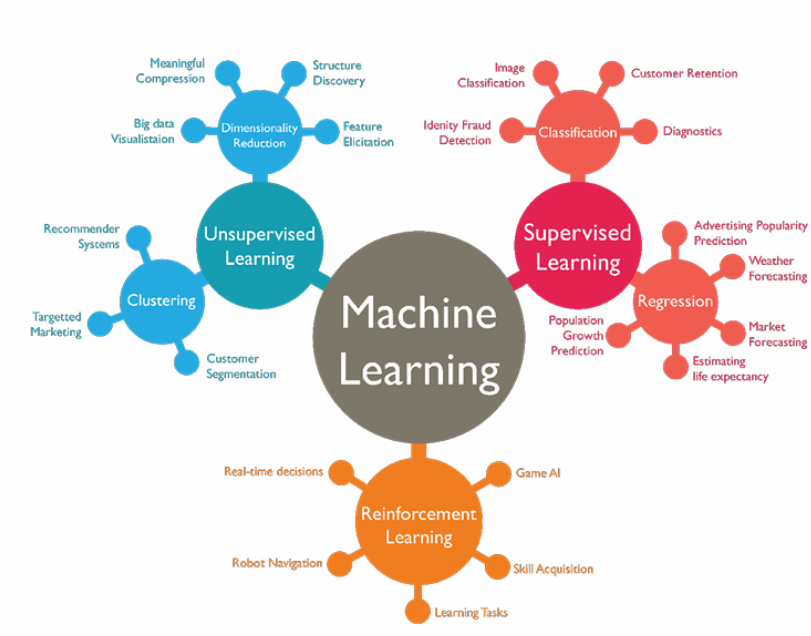
Description of the problem

1.1 Introduction

In the field of robotics, interacting with dynamic objects and performing complex tasks are fundamental challenges that require advanced engineering and artificial intelligence solutions. One of the most challenging and fascinating tasks is training a robotic arm to play ping pong. This game not only requires precise coordination between vision, movement, and strategy but also rapid reaction speed, posing a series of challenges both in hardware and software.

Ping pong is considered an ideal benchmark for testing the capabilities of robots as it combines aspects of fine motor control, real-time visual perception, and quick decision-making. The necessity to predict the ball's trajectory, position oneself correctly, and hit the ball with the right force and direction makes this task a perfect proving ground for machine learning techniques.

Our project aims to train a robotic arm to play ping pong using a combination of machine learning techniques: supervised learning and reinforcement learning. Specifically, we used a Multilayer Perceptron (MLP) for supervised training and Deep Deterministic Policy Gradient (DDPG) for reinforcement training. The MLP allows us to learn from labeled data, providing an initial base of knowledge for the robot, while DDPG refines this knowledge through continuous interaction and experiential learning.



To implement and test our model, we were provided with PyBullet as a simulation environment. PyBullet is an advanced platform for physics simulation that offers high fidelity in reproducing movements and interactions between objects. With PyBullet, we can precisely simulate the ping pong table, the ball, and the robotic arm, allowing us to conduct detailed experiments without the need for costly physical prototypes.
The project faces several technical challenges, including synchronizing visual perception with arm movement, managing reaction times, and handling the complexity of fine motor control. We utilized various technologies and software tools, including PyTorch for machine learning and specific data pre-processing libraries, to address these challenges effectively.
Our approach is innovative in that it combines supervised learning with reinforcement learning, representing an advancement over traditional methods. This combination enables the robotic arm to leverage pre-existing knowledge and continuously improve its performance through interaction with the environment.

The expected contributions of this project to the scientific and industrial community include new learning methodologies, improvements in robot control, and potential future applications. These could range from enhanced industrial automation to advanced research tools in robotics.
This report is structured as follows: we start with a detailed problem definition and the methodology used, followed by the implementation phase, and finally present the results and analysis. Through this work, we hope to contribute to the advancement of autonomous robotics and explore new learning methodologies for complex tasks. The solutions developed could have practical applications in various sectors, from industry to scientific research, opening new frontiers for the use of robots in dynamic and unstructured environments.

1.2 Motivation

Training a robotic arm to play ping pong poses significant challenges in the fields of advanced robotics and artificial intelligence. This task is not merely a technological showcase but also a pivotal opportunity to explore the current limits of robotic capabilities and drive innovation in the field.

- **Technical Challenges**: The precision required in ping pong surpasses simple mechanical movement; the robot must accurately perceive the position and velocity of the ball, predict its trajectory, and execute precise movements to hit the ball accurately. This demands a sophisticated integration of artificial vision algorithms for object recognition and control

models for accurate robotic arm movements. Managing reaction times and swiftly synchronizing perception with action are crucial for achieving competitive results in the game.

- **Practical Applications**: Beyond technical challenges, the project holds significant practical implications. A robotic arm capable of playing ping pong not only showcases its ability to tackle complex motor tasks but also finds potential applications in sectors such as industrial automation, elderly care, or physical rehabilitation, where precision and coordination are critical. Automating tasks requiring advanced motor and decision-making skills could enhance efficiency, safety, and reduce workplace injury risks in various application contexts.

- **Contribution to Research**: By implementing advanced machine learning techniques such as MLP and DDPG, the project aims to contribute to the development of new training methodologies for autonomous robots. Integrating supervised learning models with reinforcement learning not only enhances the robot's performance in ping pong but also paves the way for further research into combining diverse learning strategies for complex tasks. This approach not only improves the robot's performance in specific scenarios but also potentially broadens the applicability of advanced learning strategies across various domains of robotics and automation. Advanced research in this field could also inform the development of smarter, more adaptable robotic systems for the future.

In summary, this project is not just an academic exercise but a chance to push the boundaries of intelligent robotics and develop innovative solutions with tangible real-world impact. The ultimate goal is not only to enhance the performance of a single robot in ping pong but also to advance the field of artificial intelligence applied to robotics, opening new avenues for practical applications and significant technological improvements that could benefit multiple industrial and service sectors.

1.3 Objectives

The objectives of this project encompass several strategic goals aimed at advancing the capabilities of robotic systems in the context of playing ping pong:

1. Development of Robotic Skills: The primary objective is to train a robotic arm to achieve a high level of proficiency in playing ping pong. This includes mastering complex tasks such as real-time ball tracking, trajectory prediction, and precise motor control to effectively participate in gameplay with human-like skill.

2. Integration of Machine Learning Techniques: Another critical goal is to integrate and compare different machine learning approaches, specifically supervised learning (MLP) and reinforcement learning (DDPG). Supervised learning will provide foundational training based on labeled data, enabling the robotic arm to understand basic gameplay mechanics and strategies. Reinforcement learning, on the other hand, will allow the robot to refine its skills through continuous interaction with the environment, learning optimal strategies and improving performance over time.

3. Enhancement of Real-time Decision Making: Improving the robotic arm's ability to make real-time decisions based on visual input and environmental cues is essential. This involves optimizing algorithms for object recognition, trajectory estimation, and decision-making processes to ensure swift and accurate responses during gameplay.

4. Application and Adaptability: Beyond technical development, the project aims to explore practical applications of robotic ping pong playing. This includes investigating how such technology could be integrated into real-world scenarios such as sports training, interactive entertainment, or assistive technologies for individuals with physical disabilities. Assessing the adaptability of the robotic system to various environments and user needs is crucial for its broader applicability.

5. Contribution to Robotics Research: This project seeks to contribute new insights and methodologies to the field of robotics and artificial intelligence. By documenting the challenges encountered, the solutions implemented, and the outcomes achieved, the research aims to advance the understanding of autonomous systems and adaptive learning techniques applicable beyond ping pong to other complex tasks.

6. Validation and Benchmarking: Finally, validating the performance of the trained robotic arm against human players or predefined benchmarks is crucial. This benchmarking process will provide quantitative metrics to evaluate the effectiveness and efficiency of the developed robotic system in comparison to traditional approaches and state-of-the-art techniques in robotics and AI.

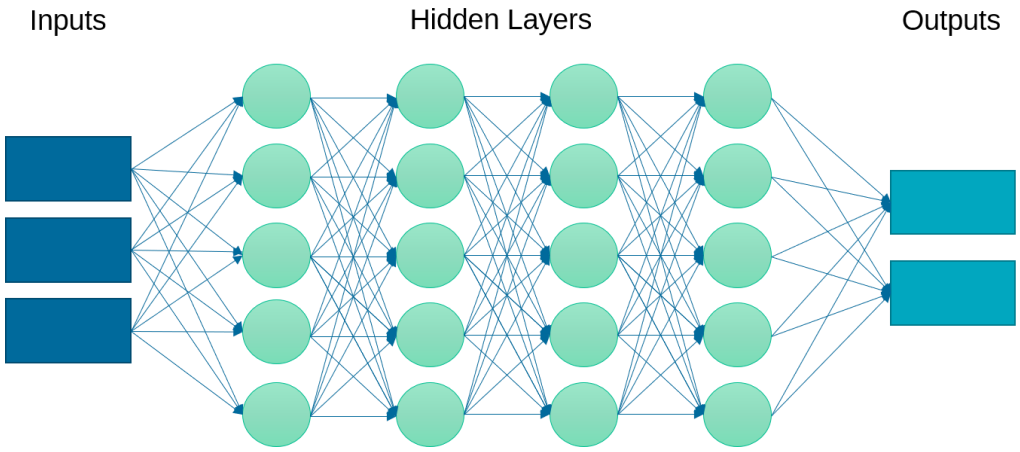
In summary, the objectives of this project are comprehensive and multidimensional, aiming to push the boundaries of robotic capabilities in complex motor tasks like playing ping pong. By achieving these goals, the project not only seeks to advance scientific knowledge but also aims to pave the way for practical applications of autonomous systems in various domains, contributing to technological innovation and societal impact.

CHAPTER 2

Solution

2.1 Machine Learning Approaches

2.1.1 Supervised Learning (MLP)



2.1.1.1 Description of Multilayer Perceptron (MLP)

The Multilayer Perceptron (MLP) represents a fundamental class of artificial neural networks characterized by multiple interconnected layers of nodes. Originating from the early developments in neural network research, MLPs evolved as a powerful model for supervised learning tasks, particularly in pattern recognition and classification.

The concept of neural networks traces its origins back to the mid-20th century, with foundational work by researchers such as Warren McCulloch and Walter Pitts in the 1940s, who proposed a computational model of artificial neurons. However, practical implementations and significant advancements in neural network research, including Multilayer Perceptrons (MLPs), emerged much later.

In the 1980s and 1990s, MLPs gained prominence due to several key developments. These included advances in computational technology that enabled the processing power necessary for training neural networks with multiple layers, and the refinement of training algorithms such as backpropagation. Backpropagation, introduced in the 1980s, revolutionized neural network training by efficiently adjusting the weights of connections between neurons based on the error gradient calculated during each iteration of training.

MLPs, with their ability to learn complex mappings between inputs and outputs through multiple layers of interconnected neurons, became a powerful tool for solving pattern recognition and classification problems. They offered significant improvements over single-layer perceptrons, allowing neural networks to capture and process more intricate patterns in data.

Since their inception, MLPs have continued to evolve alongside advancements in deep learning, becoming a foundational model in the field of artificial intelligence. They form the basis for more sophisticated neural network architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which are widely used in diverse applications ranging from computer vision and natural language processing to robotics and autonomous systems.

In summary, MLPs represent a pivotal milestone in the history of neural networks, marking a significant advancement in their capability to handle complex learning tasks and laying the groundwork for subsequent developments in deep learning and artificial intelligence.

The MLP consists of three main types of layers:

- **Input Layer**: The input layer receives raw data from the environment or dataset. Each node in this layer represents an input variable, such as the current position of the ping pong ball in the game or sensory signals from the robotic arm.
- **Hidden Layers**: Hidden layers form the core of the MLP, each composed of a variable number of nodes. Each node in a hidden layer is connected to all nodes in the previous and next layers. During the learning process, hidden layers extract and combine information from input data through linear transformations followed by non-linear activation functions, such as the Rectified Linear Unit (ReLU). This non-linearity is essential for the MLP as it allows modeling of complex relationships in data, enabling pattern representation and class discrimination.
- **Output Layer**: The output layer produces the model predictions, which depend on the specific characteristics of the problem. For example, in the context of ping pong, the output layer might generate spatial coordinates and force required for the robotic arm to successfully hit the ball. Depending on the problem type, the output layer can use different activation functions and loss criteria to optimize the model predictions.

The MLP learns from data through the process of error backpropagation, where the error is measured between the model predictions and desired output values, and this error is propagated backward

through the network to update connection weights. This iterative training allows the MLP to progressively improve its performance on the assigned task, adapting to the specific characteristics of the training data.

In the context of our project, the MLP was chosen for supervised learning phase due to its ability to generalize well to structured input data and its flexibility in adapting to the complex dynamics of ping pong gameplay. The specific configuration of the MLP used was optimized to balance model complexity with available computational resources, ensuring robust and scalable performance during simulations in the PyBullet environment.

This approach effectively enabled our system to learn from labeled data, providing a solid foundation for further refinement through reinforcement learning techniques such as Deep Deterministic Policy Gradient (DDPG), discussed subsequently.

2.1.1.2 Reason for Choosing MLP for the Problem

The selection of the Multilayer Perceptron (MLP) for this project was driven by its robust capabilities in supervised learning tasks involving structured data and its proven track record in handling complex pattern recognition challenges. Specifically tailored to training a robotic arm to play ping pong, MLPs offer several distinct advantages:

- **Capability for Complex Pattern Recognition**: MLPs are adept at learning intricate patterns and relationships within data. In the context of ping pong, where the robotic arm must interpret and respond to dynamic visual cues such as ball trajectory and opponent movements, MLPs provide a reliable framework for processing these inputs. By leveraging multiple layers of interconnected neurons, MLPs can effectively extract meaningful features from sensory inputs and translate them into precise motor actions required for gameplay.
- **Adaptability to Supervised Learning**: The supervised learning paradigm aligns perfectly with MLPs, enabling the robotic arm to learn from labeled datasets where each input (e.g., current game state) is associated with a corresponding desired output (e.g., optimal arm movement). This approach allows for systematic training iterations, where the MLP adjusts its internal parameters through backpropagation to minimize prediction errors. As a result, the robotic arm progressively refines its motor skills and strategic decision-making processes based on real-time feedback from the simulation environment.
- **Generalization to Varied Gameplay Scenarios**: MLPs are renowned for their ability to generalize knowledge learned from training data to novel, unseen situations. In the context of

ping pong, where gameplay conditions can vary widely in terms of ball speed, trajectory, and opponent strategies, the MLP's capacity to extrapolate from diverse examples ensures adaptive and consistent performance. This generalization capability is crucial for ensuring that the robotic arm can handle unforeseen challenges during gameplay without requiring extensive retraining.

- **Computational Efficiency and Scalability**: MLPs are inherently scalable and computationally efficient for tasks involving moderate-sized datasets and structured input-output mappings. The architecture of MLPs, characterized by layers of neurons with adjustable parameters, allows for flexible optimization in terms of model complexity and computational resources. This scalability is particularly advantageous in the context of simulation environments like PyBullet, where efficient utilization of computational resources is essential for iterative model training and evaluation.
- **Historical Effectiveness in Neural Networks**: Historically, MLPs have played a pivotal role in advancing the field of artificial intelligence, serving as foundational models for more sophisticated neural network architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Their enduring relevance and effectiveness in solving complex learning tasks underscore their suitability for training the robotic arm in the nuanced dynamics of ping pong gameplay.

In summary, the decision to utilize MLP for this project stems from its capabilities in handling structured data, its adaptability to supervised learning methodologies, its robust generalization capabilities, computational efficiency, and historical effectiveness in neural network research. These attributes collectively position MLP as a powerful tool for training the robotic arm to play ping pong with precision, adaptability, and efficiency in simulated environments.

2.1.1.3 Architecture and Technical Details of the Model

The Multilayer Perceptron (MLP) has been designed with a configuration aimed at effectively handling input information related to the three-dimensional position of the ball in the ping pong field, and producing outputs that accurately define the movements of the robot arm joints. Here are the detailed specifics of the architecture:

- **Input and Output**:
    - The MLP model accepts an input vector with 3 components, representing the spatial coordinates (x, y, z) of the current position of the ball. These inputs are crucial in determining the response of the robot arm in terms of positioning and movement within the simulated environment.
    - The model's output consists of a vector of size 11, corresponding to the desired movements of the robot arm joints. Each element of this vector represents critical information for the positioning and orientation of the arm during interaction with the ball in the game of ping pong.
- **Hidden Layer**:
    - The heart of the MLP is its hidden layer, designed to contain 256 neurons. Each neuron in this layer is connected to all nodes in the preceding and succeeding layers through adjustable synaptic weights. The hidden layer performs a linear transformation on the input data, followed by a non-linear activation function, such as the Rectified Linear Unit (ReLU). This choice of activation is crucial for introducing non-linearity into the layer's output, enabling the model to learn and represent complex relationships among the input data features.
- **Activation Function and Dropout**:
    - The ReLU activation function in the hidden layer was chosen for its effectiveness in facilitating rapid model convergence during the training process, while simultaneously avoiding the vanishing gradient problem. Additionally, a dropout layer with a 50% probability was incorporated to enhance the model's ability to generalize to new situations and prevent overfitting. Dropout randomly "deactivates" a percentage of neurons during training, forcing the model to not overly rely on any particular combination of neurons, thereby increasing its robustness and generalization capability.
- **Output Layer**:
    - The MLP's output layer is designed to produce an output vector of size 11, which specifies in detail the predicted movements for each joint of the robot arm. This final layer of the model applies a linear transformation to the data from the hidden layer, generating the model's final predictions for the actions of the robot arm in the specific context of the ping pong game.

- **Training and Optimization**:
    - The MLP model is trained using optimization techniques such as stochastic gradient descent, to minimize the error between the model's predictions and the desired output values. During training, the weights of the connections between neurons are updated iteratively using the backpropagation algorithm, allowing the model to gradually improve its performance in the supervised learning task.

In summary, the detailed architecture of the Multilayer Perceptron (MLP) selected for this project has been designed to maximize computational efficiency, learning capability, and generalization of the model in the dynamic and complex context of simulated ping pong.

2.1.2 Reinforcement Learning (DDPG)

2.1.2.1 Introduction to Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an advanced reinforcement learning (RL) algorithm developed to address high-dimensional continuous control problems. This approach combines elements from both Deep Q-Network (DQN) and Deterministic Policy Gradient methods, providing an effective solution for situations where actions must be taken in a continuous action space, such as the movement of a robotic arm during a game of ping pong.

The Deep Deterministic Policy Gradient (DDPG) algorithm was introduced by Timothy P. Lillicrap and colleagues in their 2015 paper titled "Continuous Control with Deep Reinforcement Learning". This work emerged from the need to extend reinforcement learning algorithms to handle continuous action spaces effectively. Traditional RL algorithms like Q-learning and its deep variant, Deep Q-Network (DQN), are designed primarily for discrete action spaces. The introduction of DDPG represented a significant advancement, addressing the limitations of previous methods when applied to problems requiring continuous control.
DDPG draws inspiration from two key algorithms in the reinforcement learning domain:

- **Deep Q-Network** (**DQN**): DQN combines Q-learning with deep neural networks, enabling the handling of high-dimensional state spaces. It uses experience replay and target networks to stabilize training, which are also employed in DDPG.

- **Deterministic Policy Gradient (DPG)**: DPG, introduced by Silver et al. in 2014, extends policy gradient methods to deterministic policies, which map states to specific actions rather than distributions over actions. This is particularly useful for continuous action spaces where the action can take any value within a range.

At its core, DDPG is an off-policy actor-critic algorithm. The main components of DDPG include the actor, the critic, experience replay, and target networks.

1. **Actor-Critic Framework**:
   - **Actor**: The actor is a neural network that determines the action to take given a state. It represents a policy, $\pi(s|\theta^{\pi})$ where s is the state and θ^{π} are the parameters of the actor network. The policy is deterministic, meaning that for each state, it outputs a specific action rather than a probability distribution over actions.
   - **Critic**: The critic is another neural network that evaluates the action taken by the actor by estimating the Q-value, $Q(s,a|\theta^{Q})$, where s is the state, a is the action, and θ^{Q} are the parameters of the critic network. The Q-value represents the expected return of taking action a in state s and following the policy thereafter.
2. **Experience Replay**:
   - Experience replay is a technique where the agent stores its experiences (s, a, r, s') in a replay buffer. During training, mini-batches of experiences are randomly sampled from this buffer to update the networks. This breaks the correlation between consecutive experiences and leads to more stable and efficient learning.
3. **Target Networks**:
   - To stabilize training, DDPG uses target networks for both the actor and the critic. These networks are copies of the original networks but their parameters are updated more slowly. This technique, known as "soft updates," helps to mitigate the problem of the moving target in reinforcement learning. The target networks are updated using the following rule:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

   where τ is a small constant (e.g., 0.001), θ are the parameters of the original network, and θ' are the parameters of the target network.

4. **Exploration and Exploitation**:
    - In the context of reinforcement learning, exploration is crucial for discovering new actions that might lead to higher rewards. DDPG addresses this issue by adding noise to the actor's deterministic policy during training. This noise can be generated using processes such as Ornstein-Uhlenbeck, which produces temporally correlated noise, suitable for continuous control environments. This approach allows the algorithm to balance exploration of new actions with the exploitation of already acquired knowledge, enhancing the overall learning effectiveness.
5. **Advantages and Applications**:
    - DDPG offers several advantages over other RL algorithms, especially for continuous control problems. Its main benefits include:
        - **Handling Continuous Actions**: The ability to operate in continuous action spaces makes DDPG ideal for applications like robotic control, where actions need to be finely tuned.
        - **Stable Convergence**: The use of target networks and soft updates contributes to improved training stability, reducing variance in Q-function estimates.
        - **Training Efficiency**: The combination of policy gradient and Q-learning techniques allows DDPG to learn effective policies with relatively fewer samples, making it suitable for applications where data collection is costly or limited.

    In the project of training a robotic arm to play ping pong, DDPG was chosen for its ability to handle continuous actions and improve control stability. The precision and effectiveness of the robotic arm's actions are crucial for interacting with the dynamic nature of the ball and the playing field, making DDPG an excellent choice for optimizing system performance.

In conclusion, Deep Deterministic Policy Gradient represents a significant advancement in reinforcement learning, providing a robust framework for addressing high-dimensional continuous control problems. Its application in the context of a robotic arm playing ping pong demonstrates DDPG's potential to enhance control and optimization capabilities in complex and dynamic environments.

2.1.2.2 Advantages of Using DDPG for Continuous Optimization

The Deep Deterministic Policy Gradient (DDPG) algorithm offers numerous advantages in the realm of continuous optimization, especially in contexts where managing actions in a continuous space is crucial. Here are the main advantages detailed extensively:

1. **Handling Continuous Action Spaces**: One of the most significant benefits of DDPG is its ability to effectively manage continuous action spaces. Traditional RL algorithms, such as Q-learning and DQN, are tailored for discrete action spaces, limiting their application to many real-world problems. In contrast, DDPG can handle actions that can take any value within a continuous range. This capability is essential for various applications, such as robotic arm movements, autonomous driving, and complex system optimizations, where actions are naturally continuous and precise control is required.
2. **Stability in Training Process**: DDPG incorporates several techniques to stabilize the training process, addressing the high variance and instability issues commonly associated with RL algorithms. It uses target networks for both the actor and the critic, which are updated slowly using a technique called soft updates. This slow update method helps reduce the divergence between the target and learned networks, thus stabilizing the learning process. Additionally, DDPG employs experience replay, which involves storing the agent's experiences and randomly sampling mini-batches for training. This approach breaks the correlation between consecutive experiences, leading to more stable and efficient learning.
3. **Efficient Exploration and Exploitation**: Balancing exploration (trying new actions) and exploitation (using known actions that yield high rewards) is crucial in RL. DDPG excels in this balance by adding noise to the actions taken by the actor network, encouraging exploration. A popular choice for this noise is the Ornstein-Uhlenbeck process, which generates temporally correlated noise suitable for physical control problems. This method allows the agent to explore a diverse range of actions, reducing the risk of getting stuck in local optima and improving the overall performance.
4. **Learning from Off-Policy Data**: DDPG is an off-policy algorithm, meaning it can learn from data generated by a different policy than the one currently being improved. This characteristic is particularly advantageous because it allows the reuse of past experiences, making the learning process more data-efficient. Agents can learn from older data, simulated experiences, or even experiences generated by other agents, significantly accelerating the training process and improving performance.

5. **Applicability to Real-World Problems**: DDPG has been successfully applied to a wide array of real-world problems requiring continuous control. For instance, in robotic manipulation, DDPG can learn to control the precise movements of a robotic arm. In autonomous driving, it can handle the continuous control needed for steering and acceleration. Furthermore, in complex system optimization, DDPG can manage and optimize continuous variables, proving its versatility and robustness across various domains.
6. **Scalability with Increasing Problem Complexity**: DDPG leverages deep neural networks to represent the actor and critic, enabling it to handle high-dimensional state and action spaces. The deep learning aspect allows DDPG to scale effectively as the complexity of the problem increases. Neural networks can learn intricate patterns and representations, which is particularly beneficial for complex, dynamic environments. As a result, DDPG can adapt to more sophisticated tasks, making it a powerful tool for advanced control problems.
7. **Long-Term Reward Optimization**: By utilizing the actor-critic framework, DDPG optimizes long-term rewards rather than short-term gains. The critic network evaluates the action chosen by the actor network by estimating the Q-value, which represents the expected cumulative reward. This approach ensures that the policy learned by the actor is geared towards maximizing long-term rewards, leading to more strategic and effective decision-making.
8. **Flexibility and Customization**: DDPG's framework allows for significant flexibility and customization. Researchers and practitioners can adjust the architecture of the actor and critic networks, modify the exploration strategy, and tune hyperparameters to suit specific problem requirements. This adaptability makes DDPG a versatile algorithm capable of being tailored to a wide range of applications.

In conclusion, DDPG offers a robust set of advantages for continuous optimization problems. Its ability to handle continuous action spaces, stabilize training, balance exploration and exploitation, learn from off-policy data, and apply to real-world problems demonstrates its power and versatility. The algorithm's scalability with increasing problem complexity, focus on long-term reward optimization, and flexibility in customization further cement DDPG as a premier choice for advanced control applications in various domains.

2.1.2.3 Architecture of DDPG: Actor and Critic Networks, and Their Interaction

In the Deep Deterministic Policy Gradient (DDPG) algorithm, the architecture consists of two primary networks: the Actor Network and the Critic Network. These networks collaborate to facilitate effective learning and decision-making in environments with continuous action spaces. Below is a detailed explanation of their architecture and interaction, emphasizing practical aspects relevant to controlling a robotic arm in a ping pong simulation.

1. **Actor Network**: The Actor Network determines the optimal actions to take based on the current state of the environment. It maps states to specific actions, crucial for controlling the robotic arm's movements.

Architecture Details:

- **Input Layer**: The Actor Network receives the current state, which includes three input nodes representing the x, y, and z coordinates of the ball's position in the ping pong simulation.
- **Hidden Layer**: The network has one hidden layer with 64 neurons. This layer uses the ReLU activation function to introduce non-linearity, enabling the network to learn complex mappings from states to actions.
- **Output Layer**: The final layer outputs the actions that control the last two joints of the robotic arm. These actions determine the movement of the paddle in response to the ball's position and trajectory. The Tanh activation function is used in the output layer to ensure the actions are within a specific range.

2. **Critic Network**: The Critic Network evaluates the actions chosen by the Actor Network by estimating the Q-value, representing the expected cumulative reward for a given state-action pair. This evaluation guides the actor towards actions that maximize long-term rewards.

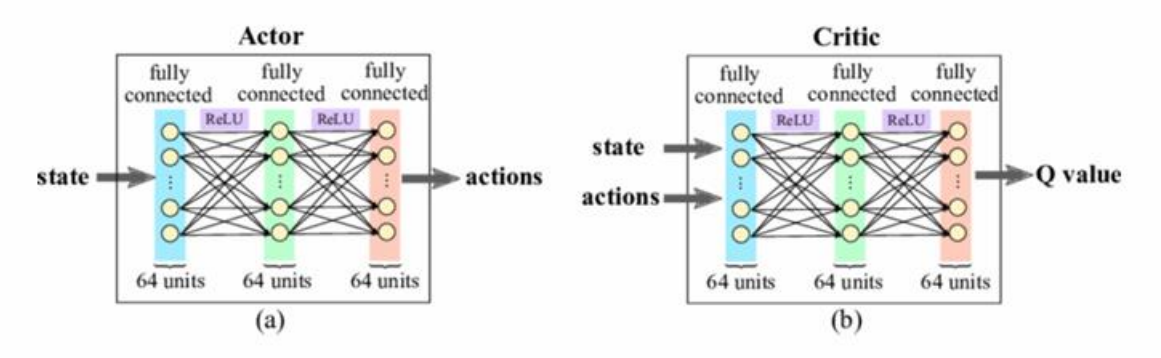
Architecture Details:

- **Input Layer**: The Critic Network takes as input the combined state and action. This input is formed by concatenating the state vector with the action vector.

- **Hidden Layer**: Similar to the Actor Network, the Critic Network has one hidden layer with 64 neurons. The ReLU activation function is applied in this layer to capture complex relationships between state-action pairs.
- **Output Layer**: The final layer outputs a single Q-value, which estimates the expected future rewards given the state and action. This value guides the actor in improving its policy to achieve higher rewards over time.

The Actor and Critic Networks collaborate in the following ways:

- **Policy Improvement** (Actor Update):
  - The Actor Network selects actions based on the current state.
  - The Critic Network evaluates these actions by estimating their Q-value.
  - The actor's parameters are adjusted using policy gradients to increase actions that lead to higher Q-values, thereby improving performance.
- **Value Estimation** (Critic Update):
  - The Critic Network learns to estimate Q-values by comparing predicted and target values based on observed rewards.
  - This estimation helps the Critic Network refine its parameters to better predict future rewards, facilitating more effective action selection by the Actor Network.
- **Experience Replay**:
  - Both networks utilize an experience replay buffer to store and sample past experiences (state, action, reward, next state).
  - This buffer breaks correlation between consecutive experiences, enhancing learning stability and efficiency.
- **Soft Updates**:
  - Target Actor and Critic Networks are periodically updated to track the main networks smoothly.
  - Soft updates ensure gradual parameter adjustments, improving the stability of training and convergence.



Hyperparameters:

- **Epsilon** (Exploration Factor): Initially set to 1.0 and decays at a rate of 0.995, encouraging exploration in the early stages and gradually reducing it to favor exploitation.
- **Gamma** (Discount Factor): Set to 0.99, it represents the degree to which future rewards are considered in the agent's decision-making process, balancing immediate rewards with long-term goals.
- **Tau** (Soft Update Factor): Set to 0.001, it controls the rate at which the target networks are updated to track the main networks, ensuring stable and gradual learning over time.
- **Batch Size**: Set to 64, determines the number of experiences sampled from the replay buffer for each training iteration, balancing between learning efficiency and computational resources.

The Actor and Critic Networks in DDPG form a robust framework for learning in continuous action spaces, such as controlling a robotic arm in a ping pong simulation. The Actor Network's role in action selection, combined with the Critic Network's evaluation and the synergy between them through policy and value updates, enables effective decision-making and performance improvement over time. This setup is ideal for complex tasks requiring precise and continuous action control, contributing to advancements in robotics and AI applications.

2.2 Training Strategy

2.2.1 Integration of Methods

2.2.1.1 Description of How MLP and DDPG are Combined

The integration strategy of Multi-layer Perceptron (MLP) and Deep Deterministic Policy Gradient (DDPG) represents a sophisticated approach to optimizing the robotic arm's proficiency in playing ping pong. This hybrid method capitalizes on the complementary strengths of both supervised learning and reinforcement learning paradigms.

Multi-layer Perceptron (MLP) initially assumes control over all 11 joints of the robotic arm. Trained via supervised learning, MLP's role is pivotal in mastering fundamental joint movements based on varying game scenarios and the current state, such as the ball's position within the simulation. MLP serves as the primary decision-maker, leveraging its trained model to predict precise joint actions that align with the current game dynamics.

Upon successful ball contact, indicated by a positive event like hitting the ball, the system seamlessly transitions to **Deep Deterministic Policy Gradient (DDPG)**. DDPG, a reinforcement learning algorithm, takes over to refine the actions of the last two joints of the robotic arm. Unlike MLP's broad control, DDPG specializes in optimizing these critical joints responsible for fine-tuned paddle movements. The focus shifts from general movement control to strategic gameplay, aiming to return the ball to the opponent's side with optimal accuracy and timing.

DDPG operates on a reward-based mechanism where successful ball contacts and effective gameplay actions yield positive rewards. This reinforcement mechanism incentivizes the robotic arm to learn and enhance its performance iteratively. By integrating MLP's predictive capabilities with DDPG's adaptive learning and goal-oriented adjustments, the hybrid approach ensures continuous refinement of gameplay strategies. This iterative learning process not only optimizes the robotic arm's responsiveness to dynamic game scenarios but also elevates its overall proficiency in competitive ping pong simulations.

Key Advantages of the Combined Approach:

- **Flexibility and Adaptability**: MLP provides a robust foundation for initial decision-making, accommodating various game scenarios and environmental conditions.
- **Precision and Optimization**: DDPG refines specific joint movements, optimizing paddle actions to maximize gameplay effectiveness and scoring potential.

- **Continuous Improvement**: The reinforcement learning loop enables ongoing learning and adjustment based on real-time feedback, enhancing the robotic arm's performance over time.
- **Synergistic Effect**: By integrating MLP's broad predictive capabilities with DDPG's focused optimization, the combined approach achieves a balanced strategy that excels in both flexibility and precision, essential for competitive ping pong gameplay.

This integrated method not only enhances the robotic arm's ability to play ping pong proficiently but also showcases the power of combining diverse machine learning techniques to tackle complex real-world challenges effectively.

2.2.1.2 Phases of the Training Process: Pre-training with Supervised Learning and Refining with Reinforcement Learning

The training process unfolds through meticulously planned phases, each essential in augmenting the robotic arm's proficiency in playing ping pong:

- **Pre-training with Supervised Learning**: Initially, the Multi-layer Perceptron (MLP) embarks on a phase of supervised learning. Trained on a comprehensive dataset encompassing diverse game scenarios and ball positions, MLP learns to predict optimal joint actions based on input states. This phase focuses on minimizing prediction errors and refining the accuracy of joint movements across varying gameplay conditions. By leveraging labeled data, MLP establishes a foundational understanding of effective joint coordination, laying the groundwork for subsequent learning phases.
- **Refining with Reinforcement Learning (DDPG)**: Upon achieving a milestone, such as successfully hitting the ball, the training regimen transitions to Deep Deterministic Policy Gradient (DDPG). This reinforcement learning algorithm specializes in fine-tuning the actions of the robotic arm's last two joints, crucial for precise paddle movements. Unlike supervised learning, DDPG operates in an interactive environment where actions are reinforced based on achieved rewards. The robotic arm dynamically adjusts its strategies in response to real-time feedback, aiming to maximize rewards by consistently hitting the ball accurately into the opponent's court. This iterative process not only optimizes joint actions but also enhances the arm's decision-making capabilities under dynamic and competitive ping pong scenarios.

By integrating supervised learning for foundational skill acquisition with reinforcement learning for adaptive refinement, the training process ensures continuous enhancement of the robotic arm's gameplay strategy. This dual-phase approach not only facilitates robust performance in ping pong simulations but also exemplifies the synergy between different machine learning techniques in tackling complex real-world challenges effectively.

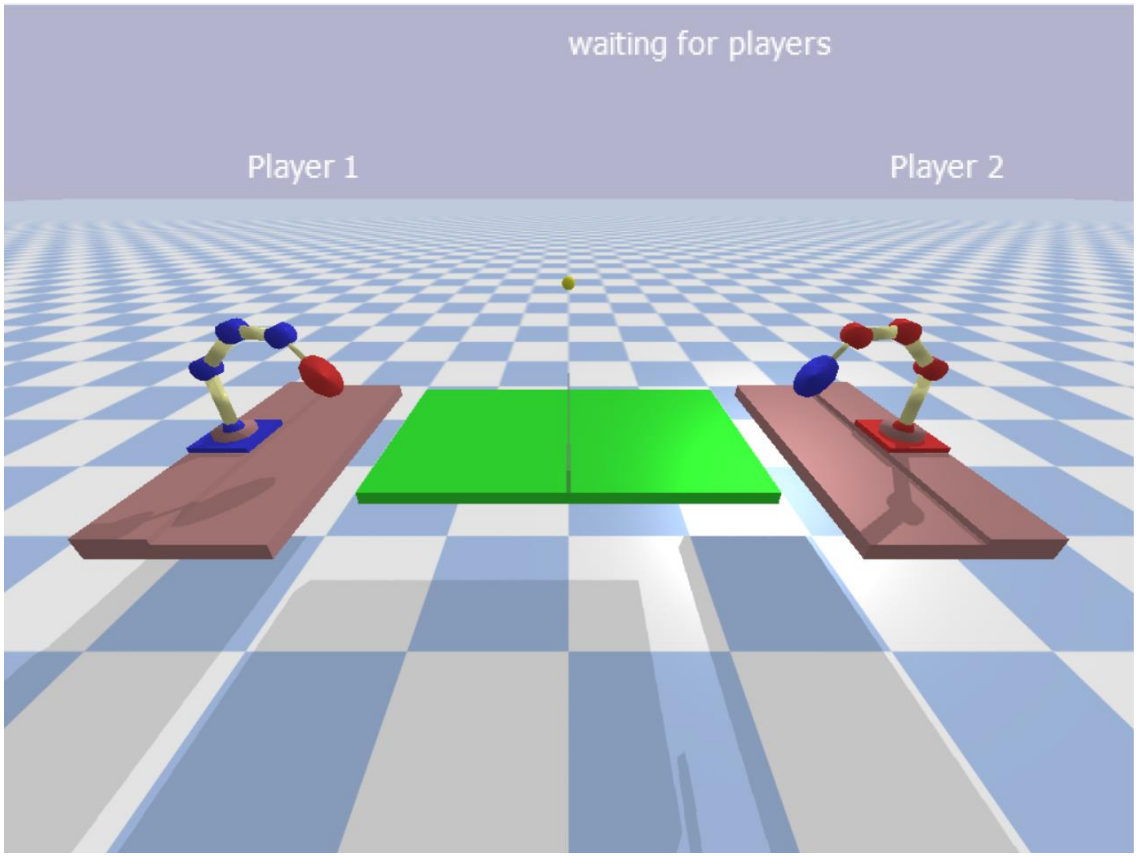
2.3 Environment Setup

- **Simulation Environment**: PyBullet is used as the physics engine to simulate complex interactions between the robotic arm, ping pong ball, and the surrounding environment. PyBullet provides a robust and realistic virtual environment, essential for developing and validating control algorithms and gameplay strategies.
- **Initial Configuration**: The ping pong table is set up according to standard specifications, with precise dimensions and a reflective surface for accurate simulation of gameplay dynamics. The ball is initially positioned within the playing field, ready to be served and played between the robotic arm and a simulated opponent.
- **Robotic Arm**: The robotic arm is modeled with joints that simulate realistic movements:
    - **Translations**:
        - Joint 0 controls forward-backward movement of the robotic arm, with values ranging from -0.3 to 0.3, where positive values indicate forward movement.
        - Joint 1 manages left-right movement, with values ranging from -0.8 to 0.8, where positive values indicate movement to the right.
    - **Rotations**:
        - Joint 2 allows rotations around the vertical axis (Z) with a full range of motion.
        - Joint 3 adjusts the pitch of the first arm link, with values from $-\pi/2$ to $\pi/2$.
        - Joints 4 and 5 control the roll and pitch of the second link, respectively, with movements up to $\pm\pi/2$.
        - Joints 6 and 7 adjust the pitch of the third arm link, with a similar range of motion.
        - Joints 8, 9, and 10 are dedicated to controlling the orientation of the ping pong paddle, allowing for pitch and roll adjustments that adhere to gameplay dynamics.

This configuration enables the robotic arm to move and react fluidly and naturally to the dynamics of the simulation, providing an ideal platform for analyzing and developing advanced gameplay strategies.

- **Game Scenarios**: Game rules are implemented to simulate competitive ping pong matches between the robotic arm and a virtual opponent. Game phases include alternating service between players, aiming to score points by hitting the ball into the opponent's half or by hitting the opponent's robot before the ball returns to one's own half.
- **Server Command Options**: The simulation server supports various configuration options to accommodate specific experiment needs:
    - -port <port>: Sets the TCP port used by the server. The default is 9543.
    - -time <limit>: Sets a time limit for the game. The limit can be specified in seconds or in the format minutes (e.g., 172 or 2:52). The default is no time limit.
    - -score <limit>: Sets a score limit for the game. The default is no score limit.
    - -noball: Plays a game where the ball is held in a fixed position, useful for acquiring data on the robot's kinematics without the ball's movement.
    - -sameserve: Plays a game where the serving player is not changed after each point, useful for learning how to consistently respond to serves.
    - -swap: Starts the service with the second player instead of the first.
    - -dummy: Adds a dummy player that only follows the x position of the ball, without actively participating in the game.
    - -auto: Adds an auto player that interacts with the simulation autonomously.
    - -nogui: Disables the graphical user interface (GUI), useful for running simulations in the background without graphical display.
    - -font <ratio>: Scales the font size used for text in the GUI by the specified ratio, allowing visual customization of the interface.

These options allow for customization of the simulation environment and player behavior to support a wide range of experiments and analyses in the context of robotic ping pong.


waiting for players
Player 1
Player 2

CHAPTER 3

Implementation

3.1 Data Collection and Preprocessing

3.1.1 Supervised Learning

3.1.1.1 Data Collection Process for Supervised Learning

The data collection process for supervised training was a critical phase in preparing the artificial intelligence model to play ping pong autonomously. The adopted methodology aimed to capture a wide range of gameplay scenarios to ensure the model was robust and capable of handling varied and realistic situations.

The dataset was constructed to include a detailed representation of all relevant states during the simulation of the robotic player arm. These states encompassed the positions and velocities of the robot's joints, the coordinates of the ball and paddle, along with various state parameters such as simulation time and game status indicators.

During the data collection phase, not only were the movements of the robot and ball recorded, but also the gameplay strategies employed in different contexts. This included defensive and offensive situations, responses to opponent shots, and variations in ball speed and trajectory. This diversity in data allowed the learning model to explore and comprehend a wide range of gameplay strategies.

A crucial aspect of data collection was detailed annotation. Each instance in the dataset was annotated with key information, such as shot outcomes (success or failure), points scored, and critical changes in game state. This provided the model with context and meaning for each data point, facilitating a deeper understanding of the flow of the game.

To ensure data quality, rigorous preprocessing techniques were applied. These included feature normalization to standardize measurement scales, handling of missing data to maintain dataset completeness, and identification and removal of outliers to ensure data integrity. These practices improved the consistency and quality of the dataset, preparing it for use in model training.

In addition to data preparation, thorough validation was conducted to ensure dataset accuracy and reliability. This process involved cross-checks between recorded data and simulations to confirm data correspondence and accuracy.

In conclusion, the obtained dataset not only provided a solid foundation for model training but also enabled the learning model to gain a detailed and robust understanding of simulated ping pong

gameplay. This targeted approach prepared the model to effectively tackle challenges and variations in the game context, promoting effective learning and continuous performance improvement.

3.1.1.2 Preprocessing Techniques Applied to Data

In the data preparation process for training the MLP model in the provided code, several preprocessing techniques were employed to ensure the data is effectively prepared for use by the model. Here is a detailed description of these techniques:

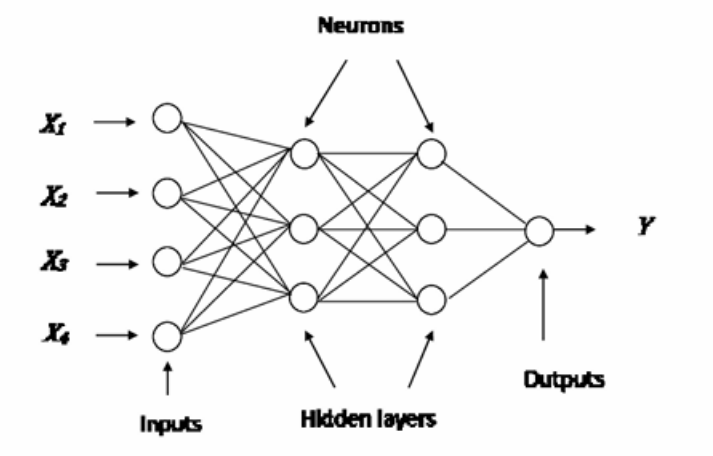
1. **Data Filtering based on Conditions**: A filtering condition was applied to remove rows where the current ball velocity (*Current ball velocity y*) equals zero. This operation was performed separately on the training, validation, and test datasets. Filtering data based on specific conditions helps to remove or handle unwanted cases that could negatively impact model training. In the context of the simulated game, removing instances where the ball is stationary helps improve the model's ability to learn ball movements during gameplay.
2. **Feature Selection**: Specific features were selected from the original datasets to be used as input for the MLP model. These features include the current positions of the ball (*Current ball position x, Current ball position y, Current ball position z*) and the current positions of the robotic arm joints (*Current joint positions 0* through *Current joint positions 10*). Selecting relevant features is crucial to ensure the model receives pertinent and meaningful information for the training task.
3. **Conversion of Data into PyTorch Tensors**: The preprocessed data was converted into PyTorch tensors of type *torch.float32*. This conversion is essential because PyTorch tensors are the standard data format used for training models in PyTorch. Using PyTorch tensors allows leveraging parallel computing capabilities on GPUs (if available), thus speeding up training and data processing.
4. **Creation of Dataset and DataLoader**: The preprocessed data was organized into *TensorDataset* and *DataLoader*. *TensorDataset* was used to wrap the input and target tensors, while *DataLoader* handled the iteration of data in mini-batches during model training. This approach optimizes computational efficiency and data management during model training.

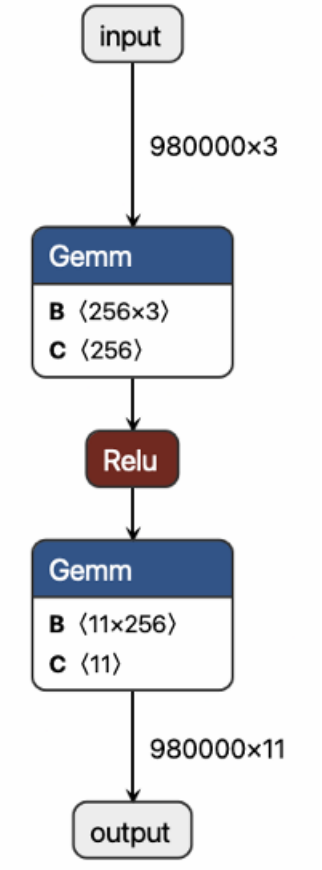
In summary, the preprocessing techniques implemented in the code aim to enhance data quality and readiness before feeding it into the MLP model. By filtering out undesired instances, selecting

relevant features, and converting data into PyTorch tensors, it ensures the model can effectively learn from the provided data, thereby improving its predictive capabilities and performance in the specific context of the simulated game.

3.2 Model Training

3.2.1 Supervised Learning





3.2.1.1 Phase of Training of the Model

During the training phase of the MLP (Multi-Layer Perceptron) model, several key steps are undertaken to optimize its ability to predict joint positions based on ball positions. Here's an overview of the training process:

1. **Data Loading and Preprocessing**:
    - The training dataset (*train.csv*) is loaded and preprocessed to ensure data quality. Rows where the ball's vertical velocity (*Current ball velocity y*) equals zero are removed to avoid biased learning.
    - Features (*X*) are extracted, including the current ball position coordinates (*Current ball position x, y, z*).
    - Target outputs (*Y*) are extracted, representing the current joint positions (*Current joint positions 0-10*).
2. **Dataset Conversion**:
    - Data from pandas DataFrames are converted into PyTorch tensors (*X_train_tensor, Y_train_tensor*) for compatibility with the PyTorch framework and to facilitate GPU acceleration if available.
3. **Data Loading with DataLoader**:
    - PyTorch's *TensorDataset* and *DataLoader* are used to handle data batching and shuffling. This setup ensures efficient loading and processing of data during training.
    - *Train_loader* batches the training dataset into manageable chunks (batch size of 64 in this case) and shuffles the data to enhance model generalization.
4. **Model Definition**:
    - An MLP model (MLP) is defined using the *nn.Module* class from PyTorch. It consists of an input layer (*input_size*), a hidden layer (*hidden_size1* with ReLU activation and dropout), and an output layer (*output_size*).
5. **Model Initialization and Optimization**:
    - The model is initialized, and if a previously trained model (*MLP_Model.pth*) exists, its weights are loaded. Otherwise, the model starts from scratch.
    - Adam optimizer is employed with a learning rate of 0.001 and weight decay (1e-5) to update the model parameters based on the computed gradients during backpropagation.

6. **Training Loop Execution**:
    - The training loop executes for a specified number of epochs (*num_epochs* = 100), iterating through the entire training dataset multiple times.
    - Within each epoch, the model is set to training mode (*model.train()*) to enable gradient calculation and parameter updates.
    - Batches of data (*data*) are fed into the model to generate predictions (*output*). The loss between predicted and actual target values (*target*) is computed using the Mean Squared Error (MSE) loss function (*criterion*).
    - Gradients are computed using *loss.backward()* and model parameters are updated via *optimizer.step()*.
7. **Validation and Early Stopping**:
    - After each epoch, the model switches to evaluation mode (*model.eval()*) to disable dropout and compute validation metrics.
    - Validation loss (*val_loss*), R-squared (*r2_val*), and MSE (*mse_val*) are calculated on a separate validation dataset (*validation.csv*) to monitor the model's performance on unseen data.
    - Early stopping mechanism is employed to prevent overfitting by monitoring validation loss. Training terminates if validation loss does not improve after a certain number of epochs (*early_stopping_patience* = 15).
8. **Model Saving**:
    - Once training completes or early stopping criteria are met, the trained model's state (*model.state_dict()*) is saved to disk (*MLP_Model.pth*).

In summary, the training phase involves iterative optimization of the MLP model's parameters using backpropagation and gradient descent. This process aims to minimize the discrepancy between predicted and actual joint positions, ensuring the model can generalize well to new data beyond the training set.

3.2.1.2 Phase of Validation of the Model

In the validation phase of training the MLP model, the goal is to assess how well the model generalizes to unseen data, which is crucial for evaluating its performance and ensuring it can make accurate predictions on new inputs.

During training, after each epoch (iteration through the entire training dataset), the model's current state is evaluated using a separate dataset called the validation set. This set contains data that the model has not seen during training and provides an unbiased assessment of its performance.

Key Steps in the Validation Phase:

- **Switching to Evaluation Mode**: Before evaluating the model on the validation set, the model is switched from training mode to evaluation mode using *model.eval()*. This step ensures that layers like dropout, which are used during training to prevent overfitting by randomly dropping neurons, are deactivated. This allows the model to produce deterministic predictions suitable for evaluation.
- **Computing Validation Loss**: The validation loss is computed as the average loss across all mini-batches in the validation set. In this case, Mean Squared Error (MSE) is used as the loss function (*nn.MSELoss()*), which measures the average squared difference between the predicted values and the actual target values. A lower validation loss indicates that the model's predictions are closer to the actual targets, reflecting better performance.
- **Calculating Additional Metrics**: Besides validation loss, additional metrics such as R-squared (R2) and Mean Squared Error (MSE) are computed. R2 score (*calculate_r2*) quantifies how well the model predicts the variation in the target variable compared to a baseline model, with higher values indicating better predictive performance. MSE (*calculate_MSE*) provides a measure of the average squared difference between predicted and actual values, highlighting the model's precision in prediction.
- **Early Stopping**: To prevent overfitting and ensure the model generalizes well to new data, an early stopping mechanism is implemented. This involves monitoring the validation loss across epochs and stopping training if the validation loss does not improve after a certain number of epochs (defined by *early_stopping_patience*). This helps in selecting the optimal model that balances training performance and generalization ability.
- **Model Saving**: After training, the best performing model on the validation set (based on validation loss) is saved. This ensures that the model state can be preserved and reused for making predictions on new data without needing to retrain from scratch.

In summary, the validation phase plays a critical role in assessing the MLP model's ability to generalize to unseen data by evaluating its performance metrics such as validation loss, R2 score, and MSE. By validating the model on independent data, it ensures robustness and reliability in its predictions, crucial for real-world applications.

3.2.1.3 Phase of Test of the Model

In the testing phase of the model, the trained MLP (Multi-Layer Perceptron) is evaluated using the separate test dataset that was not used during training or validation.
After the MLP model has been trained and validated, it needs to be tested to assess its performance on unseen data.
Key Steps in the Testing Phase:

- **Loading Test Data**: The test dataset, which consists of inputs (*XT*) and corresponding target outputs (*YT*), is loaded. This dataset was not used during training or validation and serves as an independent measure of the model's performance.
- **Preparing Test DataLoader**: The test dataset is converted into PyTorch tensors (*X_test_tensor, Y_test_tensor*) and wrapped into a *TensorDataset*. This dataset is then loaded into a *DataLoader* (*test_loader*) with a specified batch size. The *test_loader* iterates through the test dataset in batches, enabling efficient processing on the GPU if available.
- **Loading the Trained Model**: The trained MLP model is loaded from the saved state (*MLP_Model.pth*). This ensures that the model is in the same state as it was after training, with all learned weights and biases intact.
- **Evaluation Mode**: Before making predictions on the test dataset, the model is switched to evaluation mode using *model.eval()*. This step ensures that layers like dropout are deactivated, ensuring deterministic predictions suitable for evaluation.
- **Computing Test Loss**: The test loss is calculated as the average loss across all mini-batches in the test dataset. Here, Mean Squared Error (MSE) serves as the loss function (*nn.MSELoss()*), measuring the average squared difference between predicted and actual target values.
- **Calculating Additional Metrics**: Alongside test loss, additional metrics such as R-squared (R2) and Mean Squared Error (MSE) are computed. R2 score (*calculate_r2*) quantifies how well the model predicts the variation in the target variable compared to a baseline model, with

higher values indicating better predictive performance. MSE (*calculate_MSE*) provides a measure of the average squared difference between predicted and actual values, indicating the model's precision in prediction.

- **Reporting Results**: Finally, the test loss, R2 score, and MSE are printed to the console. These metrics provide insights into how well the trained MLP model generalizes to new, unseen data, and whether it meets the performance criteria set during training and validation.

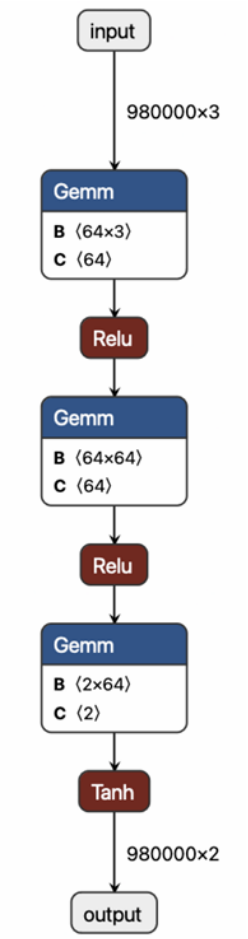
The results are:

Test Loss: 0.017454247448164747,
Test R2: 0.38290219713094464%,
Test MSE: 0.017468770796133817

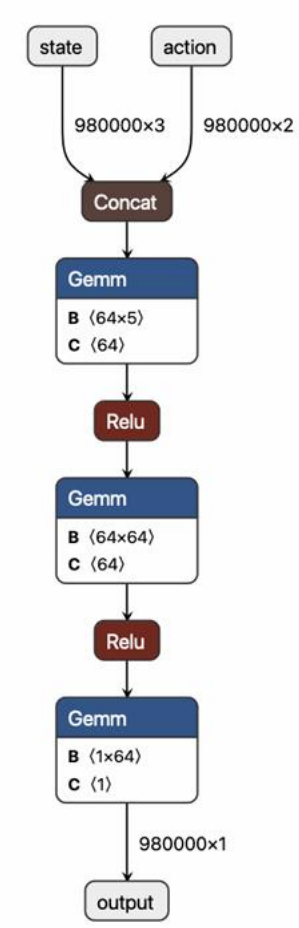
In summary, the testing phase is crucial for evaluating the MLP model’s performance on independent data, ensuring that it can make accurate predictions on new inputs beyond the training set. This step is essential for assessing the model’s readiness for deployment in real-world applications.

3.2.2 Integration of the MLP into the DDPG framework and refining process

ActorNetwork



CriticNetwork

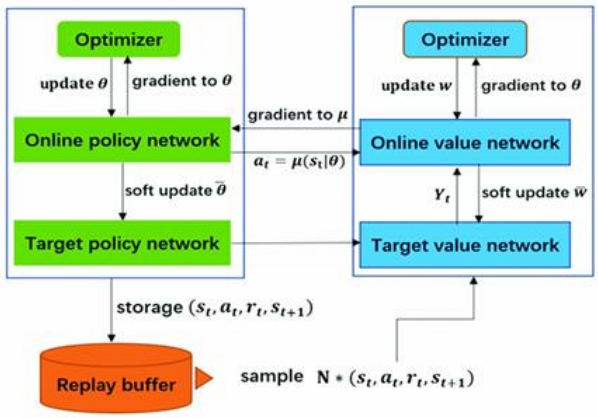


Integrating the MLP into the DDPG (Deep Deterministic Policy Gradient) framework represents a sophisticated approach to enhancing the robotic arm's gameplay capabilities. In this integration, the MLP assumes responsibility for controlling the initial nine joints (from joint 0 to joint 8) of the robotic arm. Its primary function is to process real-time data, particularly the coordinates of the ball in the game environment. Using this information, the MLP calculates optimal joint positions and movements to intercept and interact with the ball effectively.

The MLP's role is crucial in preemptively positioning the arm based on predictive models derived from the ball's trajectory and current position. By continuously adjusting the joint angles, the MLP aims to maintain optimal spatial alignment for successful ball interception and gameplay maneuvers. This predictive capability not only enhances the robotic arm's precision but also ensures timely and accurate responses to dynamic changes in the game scenario.

Concurrently, the DDPG framework governs the control of the remaining two joints (joint 9 and joint 10) of the robotic arm. Its decision-making process is contingent upon specific conditions, particularly

activated when the ball contacts the paddle. This conditional control mechanism allows the DDPG to refine and adjust the robotic arm's actions based on real-time interactions observed during gameplay. The collaborative nature of this integration ensures that the robotic arm operates synergistically with both predictive accuracy (MLP) and adaptive responsiveness (DDPG). By leveraging MLP's predictive modeling and DDPG's adaptive control strategies, the integrated system optimizes the arm's overall performance. This approach not only enhances the arm's ability to engage dynamically with the game elements but also improves its efficiency in achieving gameplay objectives effectively. Thus, through the combined strengths of MLP and DDPG, the robotic arm can navigate complex game dynamics and enhance its gameplay proficiency significantly.



3.2.3 Technical Details on DDPG Training

Training the Deep Deterministic Policy Gradient (DDPG) framework was an intensive process that took 5 days to run, mainly due to the limited computational resources available. During this time, several crucial hyperparameters were optimized to maximize the agent's performance in the gaming environment.

- **Hyperparameter Tuning**: Hyperparameter tuning was carefully performed to ensure that the DDPG model was optimally configured for the robot joint control task. The following parameters have been adjusted:
    - **Learning Rate**: Optimized to balance learning speed with training stability. Too high a value could have caused instability, while too low a value would have further prolonged the training time.

- **Batch Size**: Increased to improve training stability, although this resulted in more memory intensive use. A larger batch size helped reduce noise in the gradients and improve the convergence of the neural networks.
- **Gamma** (Discount Factor): Configured to balance the importance of future rewards versus immediate ones. A higher range was chosen to promote long-term policy learning, which is essential for success in complex control challenges.
- **Tau** (Soft Update Parameter): Adjusted to control the update speed of target networks to core networks. A smaller tau value favored more graduated updates, improving long-term training stability.
- **Noise Parameters** (OU Noise): The OU noise parameters have been calibrated to balance agent exploration with the need to converge towards optimal action policies. The OU Noise was essential to stimulate the agent's exploration in a way consistent with the game environment.

- **Duration of Training**: The entire training process took a total of 5 days to run due to limitations of available computational resources. During this period, the agent continued to interact with the environment, accumulating experiences in the replay buffer and updating the actor and critic neural networks. The long duration was necessary to ensure stable and reliable convergence of the agent's action policies.
- **Exploration Policies**: The exploration policies implemented in the DDPG framework were critical to ensuring that the agent effectively explored the space of available actions. OU Noise was used to add a stochastic exploration component to the actor's selected actions, facilitating the agent's exploration in a way that maintained coherence with the game environment. This approach helped the agent discover and learn optimal action strategies over the course of prolonged training.

Despite the challenges imposed by limited computational resources, training of the DDPG framework was a success, producing robust action policies and advanced control capabilities for the robot's joints. The detailed approach to tuning hyperparameters, managing training duration, and implementing effective exploration policies ensured that the agent could dynamically adapt to changes in the game environment. This result demonstrates the effectiveness of DDPG in solving complex control tasks through deep and reinforcement-based learning.