Quantum-Enhanced Reinforcement Learning Using Causal Models to Improve Resilience under Observational Interference

A thesis
Submitted towards the partial fullfilment of
BS-MS dual degree programme
by

Namasi G. Sankar

Date: 27-03-2024

under the guidance of

Dr. M Girish Chandra

TCS Research

from May 2023 to Mar 2024

Indian Institute of Science Education and Research
Pune

Certificate

This is to certify that this dissertation entitled ”Quantum-Enhanced Reinforcement Learning Using Causal Models to Improve Resilience under Observational Interference” submitted towards the partial fulfillment of the BS-MS degree at the Indian Institute of Science Education and Research, Pune represents original research carried out by “Namasi G. Sankar” at “TCS Research”, under the supervision of “Dr. M. Girish Chandra” during academic year May 2023 to March 2024.

[illegible]

Namasi

Supervisor:
Dr. M. Girish Chandra
Principal Scientist TCS Research

Namasivayam Gomathi Sankar
20191026
BS-MS
IISER Pune

Date:
27/03/2024

Declaration

I, hereby declare that the matter embodied in the report titled "Quantum-Enhanced Reinforcement Learning Using Causal Models to Improve Resilience under Observational Interference" is the results of the investigations carried out by me at the "TCS Research" from the period 08-05-2023 to 27-03-2024 under the supervision of Dr. M. Girish Chandra and concurrent co-supervision of Dr. Leelavati Narlikar and the same has not been submitted elsewhere for any other degree. Wherever others contribute, every effort is made to indicate this clearly, with due reference to the literature and acknowledgment of collaborative research and discussions.

[illegible]

Supervisor:
Dr. M. Girish Chandra
Principal Scientist TCS Research

[illegible]

Namasivayam Gomathi Sankar
20191026
BS-MS
IISER Pune

Date:
27/03/2024

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. M Girish Chandra, for his guidance, support, and invaluable feedback throughout the course of this research. His expertise and encouragement have been instrumental in shaping this thesis.

I am also thankful to my co-supervisor, Dr. Leelavat Narlikar and expert, Dr. Bijay Kumar Agarwalla, for their consistent support during the research. I am extremely grateful to Ankit Khandelwal, for helping me with brainstorming, understanding the details of the research, assisting me with troubleshooting and mentoring me throughout the period of the research.

Furthermore, I extend my appreciation to my friends and family for their unwavering support and encouragement during this journey.

Abstract

Deep Reinforcement Learning (DRL) is a sub-field of Machine Learning (ML) that combines reinforcement learning (RL) with deep learning techniques. DRL has showcased promising results in gaming environments as well real-world applications such as self driving vehicles and even natural language processing. However, DRL agents are susceptible to faulty observations due to sudden interference like blackouts, frozen observation, or adversarial interference in practical applications. These scenarios can hamper the learning and performance of DRL agents if they are not resilient. Drawing inspiration from causal inference, the Causal Inference Architecture (CIA) explores a Deep Q-Network (DQN) and a Proximal Policy Optimization (PPO) framework that undergoes training with an additional task focusing on training for observational interference. Through an evaluation conducted in the gymnasium Cartpole-v1 environment, the experimental findings demonstrate that the CIA exhibits enhanced performance and greater resilience against observational interference in both DQN and PPO implementations.

The CIA is further explored using quantum networks and hybrid quantum-classical networks and are tested for their resilience in noisy environments. A novel Quantum-Enhanced CIA (QCIA) is proposed which could replicate the functioning of the CIA using Variational Quantum Circuits (VQCs) and is benchmarked against the corresponding classical architectures for its performance using both DQN and PPO implementations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With recent advances in machine learning (ML) and Artificial Intelligence (AI), we have seen the benefits we can reap from this technology. AI is already embedded in our daily lives, from digital personal assistants to recommendation systems and even autonomous vehicles (1). However, current AI models have limitations, including the computational power required for training and the handling of complex tasks (2).

On the other hand, quantum computing stands at the brink of a new era of computation, promising to unlock unprecedented computational power for certain tasks.(3). Quantum computing harnesses the principles of quantum mechanics to process information in a manner fundamentally different from classical computing (4). Quantum computing is currently in the Noisy Intermediate-Scale Quantum (NISQ) era (5), where significant limitations constrain computational capabilities. The presence of noise and errors in current quantum devices hampers the accuracy of calculations (5). The lack of full-fledged quantum error correction makes it challenging to mitigate the impact of these errors. As a result, the NISQ era primarily focuses on near-term applications that can leverage the limited capabilities of these devices.

This thesis specifically delves into the exploration of Quantum-Enhanced RL (QRL). RL involves agents learning policies and making decisions based on a state space provided to them.

In the context of real-world applications, agents utilizing DRL face numerous challenges. The primary issues arise from the environment's unpredictable and often disruptive nature, which introduces uncertainties that can significantly impact an agent's performance and reliability (6). Real-world scenarios involve inherent observational uncertainty, which could be caused by external attackers or noisy sensors. This uncertainty can lead to faulty observations, disrupting the learning process and impacting the decision-making ability of the DRL agent. For example, in an autonomous driving scenario, various unforeseen events such as sudden blackouts, frame-skipping due to network instabilities, or temporary blindness while driving, can cause abrupt interference in the observations received by the DRL agent. These interruptions can lead to serious issues for the agent's learning and decision-making processes.

DRL agents need to be resilient against these interferences. They must be able to diagnose observations accurately, filter out irrelevant disruptions, and make correct inferences about reward information despite the interference. The CIQ framework in Ref. (6) is designed to train RL agents to effectively handle unforeseen and potentially catastrophic interference by incorporating observational disruptions as an auxiliary task during training. This involves providing interference labels during training and guiding the RL agent to learn to discern and adapt to interference. Specifically, the RL agent is trained to recognize interference labels using uncertain noise generators, enabling it to embed a latent state into its model, preparing it to handle such interference during deployment in the field.

To bolster an RL agent's resilience against interference, utilizing causal inference models can help the agent reason about desired rewards despite intermittent disturbances (6). The RL agent can focus on relevant reward information associated with an unobserved confounder, allowing it to make informed decisions in the presence of disruptions. The CIQ algorithm leverages causal inference on state variables and employs treatment switching methods to embed latent variables into the agent's decision-making process. CIQ uses the learned model to estimate the latent state and interference label during testing, enabling the RL agent to navigate and make decisions despite observational disruptions. The amalgamation of latent states through causal inference enhances the Q-network's ability to collect rewards in challenging environments affected by observational interference.

This thesis explores an adaptation of the CIQ called Causal Inference Architecture (CIA). This architecture is easily adapted to both DQN (7) and PPO algorithms. It further explores quantum processing in certain or all the modules of the CIA.

VQC's (8) are employed for the agent's networks and test their performance in the Gymnasium (9) CartPole-v1 environment.

The results of this paper show the feasibility of VQCs and Hybrid Quantum-Classical networks to provide competent resilience and performance as classical networks, when utilizing much lower number of parameters in the model.

A significant part of the research presented here has been published as a part of the COMSNETS 2024 conference, which can be referred to from (10)

Chapter 2

Background

2.1 Machine Learning

ML makes use of statistical techniques and computational algorithms to analyze data, make predictions and model complicated functions without relying heavily on explicit programming instructions. It instead allows algorithms to learn purely from past data or experience. The algorithm autonomously learns the most effective approach to solving the problem by trying to optimize some objective function, representing the task.

ML techniques come under three broad categories (11):

Supervised Learning: These algorithms aim to learn from labeled data, where each input is associated with a corresponding "correct" output label. They analyzing patterns in the labeled data and learn to predict or classify new, unseen data instances.

Unsupervised Learning: These algorithms operate on unlabeled data, aimed at trying to discover classifications and structures within the data. Without explicit guidance, the algorithms aim to identify groupings among data points, enabling tasks such as clustering or dimensionality reduction.

Reinforcement Learning: RL algorithms aim to train an agent to make sequential decisions in an interactive environment. The environment provides feedback in the form of rewards or penalties to the agents as they navigate the environment. Through trial and error, reinforcement learning algorithms learn optimal strategies to maximize cumulative rewards over time.

The versatility of ML techniques allow them to address a variety of challenges, ranging from image recognition and natural language processing to financial forecasting and robotics.

2.1.1 Supervised vs. Unsupervised Learning

Supervised learning constitutes a category of ML dedicated to deriving a function that effectively addresses labelled problems. It operates on a designated training set, comprised of data points $\{(x_i, y(x_i))\}$, where x_i represents input values and $y_i = y(x_i)$ denotes the corresponding expected output (11). This training set serves as a reference for evaluating the efficacy of the method. When the scores on the training data reach a satisfactory threshold, the model is considered ready to be deployed to generate predictions for previously unseen data.

In contrast, unsupervised learning focuses on identifying patterns within unstructured data. Common objectives include grouping or classification tasks wherein data points exhibiting similar characteristics, as determined by a specified metric, are clustered together. Conversely, dissimilar points are assigned to distinct groups. Unsupervised learning usually aims to minimize a specific cost function, like the sum of the Euclidean distance between points in clusters.

2.2 Reinforcement Learning

RL is an ML algorithm that aims to train networks by making use of positive and negative feedback. In an RL problem, an agent navigates an unknown environment and needs to choose suitable actions at each step to manipulate and interact with the environment. The goal of the agent is to map the possible states in an environment to possible actions, using an optimized policy, to maximize a cumulative reward signal from the environment. The policy serves as a strategy for the agent to navigate the environment. RL is commonly used for autonomous systems in dynamic environments such as robotics, game agents, and even self driving vehicles.

The different aspects in RL include (12):

Environment: The external system with which an agent interacts. The environment provides feedback to the agent based on the actions taken, influencing the subsequent state and rewards.

State: A state is usually represented as a vector that describes information about the current configuration that the environment is in. The state is crucial because the agent's policy is often based on its perception of the current state.

Agent: The agent can observe the state of the environment and take actions to interact with it. The agent takes actions in the environment based on its current knowledge or policy and tries to learn how to maximize rewards.

Action: The set of possible moves that the agent can make. The agent actions, in turn, influence the state of the environment.

Policy: A strategy that the agent follows to determine its actions. The policy can be deterministic, mapping states to specific actions, or stochastic.

Reward: The environment returns a reward value for actions taken by the agent, based on the current state of the environment. The reward signal is used to guide the agent's learning, indicating the desirability of the action taken in the given state.

Reward Function: A mapping that defines the reward an agent receives for each possible state-action pair.

Value Function: An estimation of the expected cumulative reward that an agent can obtain from a given state.

Exploration: Exploration involves trying new actions to collect data on their effects on the environment and learn strategies.

Exploration: Exploitation involves selecting actions based on knowledge from previous data to obtain high rewards based on the agent's policy.

Episode: A single run of interactions between the agent and the environment. An episode typically starts with the initial state, involves a series of actions, and ends when a termination condition is met.

RL faces unique challenges compared to the other paradigms ML algorithms in significant ways (13):

1. Training data is collected by the agent dynamically as it learns the environment, in an unsupervised manner, instead of being curated beforehand.

2. Actions taken now may have long term consequences and reception of feedback for that action is delayed. Additionally, the goal of the agent

is not to maximize the reward for a decision taken in the current state of the environment, but to maximize the cumulative reward earned in the episode.

3. Non-stationarity: Data collected at different times during the training procedure will not be I.I.D (Independently Identically Distributed). The agents actions determine the training data it gets access to (14).

RL has emerged as a powerful paradigm in ML, showcasing remarkable achievements across various domains. In particular, RL has revolutionized game-playing, demonstrated by AlphaGo Zero's performance through self-play (15). Recent advancements include DeepMind's DeepNash agent achieving competitive levels in Stratego and Meta's Cicero agent attaining human-level performance in Diplomacy (16), leveraging RL and language modeling techniques (17).

Furthermore, RL has found applications beyond gaming, in diverse industries such as quantitative finance (18), self-driving car technology (19), and computer hardware design (20). Notably, OpenAI's ChatGPT (21) model utilizes RL, specifically Proximal Policy Optimization (PPO) with human feedback, to optimize language generation quality, highlighting the versatility of RL methodologies.

In framing RL problems, two primary approaches are commonly employed: model-based and model-free RL. Model-based RL involves learning a representation of the environment's dynamics and state transitions, distinct from action strategy learning. Conversely, model-free RL agents focus only on learning optimal actions without explicit modeling of environment dynamics.

RL methodologies are further categorized into online and offline methods. Online RL involves incremental updates to the model as agents interact with the environment and observe outcomes. In contrast, offline RL leverages external datasets to approximate functions without direct interaction with the environment, offering advantages in scenarios with limited interaction capability (22).

Additionally, RL techniques vary between value-based and policy-based methods. In value-based RL, agents estimate discounted future rewards for actions at a given state, optimizing actions to maximize expected future rewards. Conversely, policy-based RL directly models a probability distribution over possible actions at each state, essentially capturing the policy itself.

2.3 Postulates of Quantum Mechanics

Quantum mechanics aims to provide a mathematical foundation that describes the behavior of particles in the size scales of electrons. The theory is based on a set of postulates that define the mathematical framework governing the quantum world (23).

2.3.1 Postulate 1: State of a Quantum System

The state of a quantum system is described by a vector in a complex vector space known as a Hilbert space. This vector is called the state vector or wavefunction, often denoted by $|\psi\rangle$ in the bra-ket notation. The state vector contains all the information about the system's physical properties, including position, momentum, and spin.

2.3.2 Postulate 2: Observables and Operators

Observable quantities, such as position, momentum, and energy, are represented by linear, Hermitian operators in quantum mechanics. Operators act on the state vector to extract information about the corresponding observable. For example, the position operator $\hat{x}$ operates on the state vector to yield the position of the particle.

2.3.3 Postulate 3: Measurement and Probability

When an observable is measured in quantum mechanics, the outcome is probabilistic rather than deterministic. The probability of obtaining a particular measurement outcome is given by the square of the absolute value of the inner product between the state vector and the eigenvector corresponding to that outcome. Mathematically, for an observable represented by the operator $\hat{A}$ with eigenvalues a_i and corresponding eigenvectors $|a_i\rangle$, the probability of measuring the outcome a_i when the system is in state $|\psi\rangle$ is given by $p(a_i) = |\langle a_i|\psi\rangle|^2$.

2.3.4 Postulate 4: Expected Values

The expected value of any operator $\hat{A}$ is given by

$$\langle A \rangle = \frac{\int_{-\infty}^{\infty} \Psi^* \hat{A} \Psi d\tau}{\int_{-\infty}^{\infty} \Psi^* \Psi d\tau}.$$

2.3.5 Postulate 5: Evolution of Quantum Systems

The time evolution of a quantum system is governed by the Schrödinger equation, which describes how the state vector changes over time. The Schrödinger equation is given by $i\hbar\frac{\partial}{\partial t}|\psi\rangle = \hat{H}|\psi\rangle$, where $\hbar$ is the reduced Planck constant, $\frac{\partial}{\partial t}$ is the partial derivative with respect to time, $\hat{H}$ is the Hamiltonian operator representing the total energy of the system, and i is the imaginary unit.

2.3.6 Postulate 6: Antisymmetry principle

The complete wavefunction is antisymmetric to the interchange of the coordinates of two particles. The intrinsic "spin" is also a part of these coordinates. The Pauli exclusion principle arises from this postulate.

2.4 Quantum Computing

2.4.1 The Qubit

A qubit is a quantum system in a two-dimensional state space. In the computational basis, it is defined as the superposition

$$|\psi\rangle = a|0\rangle + b|1\rangle.$$

A qubit can be visualized on a Bloch sphere. The coefficients a and b must be normalized such that $|a|^2 + |b|^2 = 1$. From the Euler formula $e^{ix} = \cos(x) + i\sin(x)$, we can rewrite the equation, as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle,$$

where the real numbers θ and ϕ define a point $(\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$ on the Bloch sphere as shown in Fig. 2.1.

2.4.2 Quantum Gates

Quantum gates are fundamental building blocks in quantum computing that operate on qubits, the basic units of quantum information. These gates are analogous to classical logic gates used in classical computing but have unique properties due to the principles of quantum mechanics.

In quantum computing, gates are represented by unitary matrices that act on the state of qubits. A unitary matrix preserves the length of the quantum state vector, ensuring that the probabilities of all possible outcomes sum to one. This property is essential for maintaining the principles of quantum mechanics, such as superposition and entanglement.

The RX, RY, and CZ gates are examples of commonly used quantum gates which are used in this research (24):

1. **RX Gate (Rotation around the X-axis)**: The RX gate rotates the qubit state vector around the X-axis of the Bloch sphere by a specified angle. It is expressed as:

$$RX(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$$

Here, θ is the rotation angle. The RX gate allows the qubit to be rotated between the computational basis states $|0\rangle$ and $|1\rangle$, enabling the manipulation of phase and amplitude.

2. **RY Gate (Rotation around the Y-axis)**: Similar to the RX gate, the RY gate rotates the qubit state vector, but this time around the Y-axis of the Bloch sphere. It is expressed as:

$$RY(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$$

The RY gate is particularly useful for manipulating the probability amplitudes of the qubit states, allowing for the creation of superposition states.

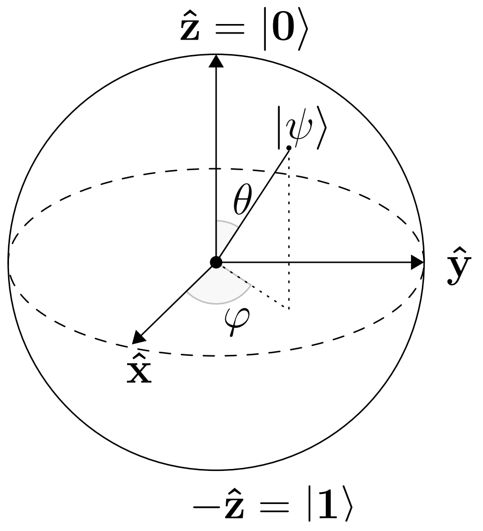
3. **CZ Gate (Controlled-Z Gate)**: The CZ gate is a two-qubit gate that performs a conditional phase shift on the target qubit depending on the state of the control qubit. It is expressed as:

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

When the control qubit is in state $|1\rangle$, the CZ gate applies a phase shift of $-\pi$ to the target qubit's state $|1\rangle$. Otherwise, it leaves the target qubit's state unchanged. The CZ gate is often used in quantum circuits for entangling qubits and implementing quantum algorithms.

2.4.3 Quantum Circuits

Qubits are the unit of information in quantum computing like the classical bit in classical computers. However, a significant difference is that classical



$\hat{\mathbf{z}} = |\mathbf{0}\rangle$
$|\psi\rangle$
θ
$\hat{\mathbf{y}}$
φ
$\hat{\mathbf{x}}$
$-\hat{\mathbf{z}} = |\mathbf{1}\rangle$

Figure 2.1: Bloch sphere representation of a single qubit. (14)

bits can only be in either of the binary states - 0 or 1 but qubits can be in a superposition of two states (14). It can represented as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix},$$

where α and β are two complex numbers and $|0\rangle$ and $|1\rangle$ are the two states in the computational basis. After a measurement, the probability that the qubit's wavefunction collapses to state $|0\rangle$ is $P(|0\rangle) = |\alpha|^2$ and to state $|1\rangle$ is $P(|1\rangle) = |\beta|^2$.

When the qubit remains unobserved, it is in a superposition of these basis states. We represent the state of the qubit diagrammatically, on the surface of the Bloch sphere:

Quantum operators are linear maps from a Hilbert space to itself and are represented using unitary matrices. Given a Hilbert space H, an operator U is a linear map $H \to H$ such that:

$$UU^\dagger = U^\dagger U = I,$$

where $U^\dagger$ is the Hermitian conjugate of U.

Gates are amalgamation of operators performed in series on qubits. We can put these gates together to form quantum circuits. Gates can act on multiple qubits at a time, resulting in a linear transformation. Since quantum operators must be unitary functions, the composition of these functions is also unitary.

There are many similarities between NNs and quantum circuits. They are both a set of operations acting sequentially on inputs, and produce measurable outputs.

Quantum machine learning (QML) makes use of quantum circuits for ML with the potential to revolutionize both fields. QML aims to leverage the principles of quantum mechanics to enhance traditional machine learning algorithms and solve computational problems more efficiently.

One of the key promises of QML lies in its ability to harness superposition and entanglement to process and manipulate data in ways that classical computers cannot. Quantum computers can represent and manipulate vast amounts of information simultaneously, offering exponential speedup for certain algorithms compared to classical counterparts.

Several approaches to QML have emerged, each leveraging different aspects of quantum computing. Quantum-enhanced algorithms, such as quantum support vector machines (25) and quantum neural networks, aim to improve the performance of classical machine learning tasks by exploiting quantum parallelism and interference. These algorithms hold promise for tasks such as optimization, classification, and pattern recognition.

Another approach is to use quantum computers to directly implement and execute classical machine learning algorithms, albeit with potential performance gains due to quantum parallelism. Quantum annealing and adiabatic quantum computing are examples of this approach, focusing on solving optimization problems efficiently (26).

Despite its promise, QML is still in its early stages, facing significant challenges and limitations. One major challenge is the development of quantum hardware with sufficient coherence times and qubit fidelity to support complex QML algorithms. Additionally, the lack of standardized software tools and programming frameworks hinders the widespread adoption and development of QML algorithms.

2.5 Causal Modelling

Causal modeling, a field drawing from disciplines like econometrics, epidemiology, and computer science, focuses on understanding causal relationships within systems.

A causal model is a mathematical representation of these causal relationships, encapsulating the causal structure of a system. This structure enables us to predict the outcomes of interventions, allowing us to understand how changing certain variables affects the overall system behavior.

Unlike correlation, which merely identifies associations between variables, causal modeling delves deeper into the mechanisms behind these associations. By uncovering causal relations, we gain insights into how variables interact and influence each other.

Causal models consist of three key components: the system itself, the probabilistic distribution of the system's variables, and the causal graph representing the causal relationships among variables. The causal graph, composed of nodes and directed arrows, illustrates which variables directly influence others.

In reinforcement learning, incorporating causal graphs can provide valuable guidance for decision-making. By understanding the causal relationships between actions and outcomes, agents can make more informed decisions and adapt to changing environments more effectively. Additionally, causal modeling can help identify confounding variables and mitigate bias in the learning process.

Moreover, causal modeling doesn't just provide insights into observed data; it also allows for hypothetical interventions. By simulating the effects of potential interventions, we can anticipate how changes to the system will impact future outcomes.

Reinforcement learning has seen considerable progress and excitement in recent years. It holds promise for further advancement by embracing causal techniques. In online reinforcement learning, there is an inherent causal structure, wherein the environment responds to the agent's actions, and the agent learns from these outcomes. Causal reinforcement learning introduces causal information, such as causal graphs and models, into the standard reinforcement learning framework.

One significant advantage of causal inference is its capability to integrate domain knowledge. This aspect makes causality particularly attractive for reinforcement learning, where enhancing agent performance can be achieved by incorporating external knowledge about the environment. For example, in healthcare applications of reinforcement learning, prior knowledge about drug interactions, commonly known to medical professionals, can be encoded using causal methods.

Moreover, two key challenges in reinforcement learning are sample efficiency (the number of episodes needed for an agent to achieve satisfactory performance) and the effective utilization of offline data (data collected by observing another agent in a similar system) in an online learning scenario. As discussed later, researchers in causal reinforcement learning are leveraging causal techniques to address these challenges.

Chapter 3

Methods

3.1 Neural Networks

A neural network is a computational model inspired by the structure and function of the human brain's neural networks. It consists of interconnected nodes, called neurons, organized into layers. These neurons receive input, perform computations, and produce output signals.

In a typical neural network, there are three main types of layers:

Input Layer: receives the initial input data, which could be images, text, or numerical values. Each neuron in the input layer represents a feature or attribute of the input data.

Hidden Layers: These are intermediary layers between the input and output layers. Each neuron in a hidden layer takes input from the neurons in the previous layer, performs a computation using weighted connections, and passes the result to the neurons in the next layer.

Output Layer: The output layer produces the final output of the neural network, which could be a classification label, a regression value, or some other form of prediction. The number of neurons in the output layer depends on the nature of the task the neural network is designed to perform.

Neural networks need to first learn from training data approximate patterns and relationships in the input data into functions. During training, the network adjusts its weights and biases based on the input-output pairs provided in a training dataset. This adjustment process, often referred to as backpropagation, aims to minimize the difference between the network's predictions and the actual outputs.

Neural networks use activation functions to introduce non-linearities into the model (27). Without activation functions, the network would essentially be a linear regression model, which can only learn linear relationships between input and output.

1. **Linear Transformations**: In a neural network layer, each neuron performs a linear transformation of the input data by computing the dot product of the input vector with the weights and adding a bias term.

2. **Activation Functions**: After the linear transformation, the output is passed through an activation function. Activation functions introduce non-linearities by applying a non-linear transformation to the input. This allows the network to learn complex patterns and relationships in the data.

3. **Non-linear Mapping**: The non-linearities introduced by activation functions enable the neural network to learn non-linear mappings from the input data to the output. This is crucial for capturing the complex relationships that exist in real-world data, such as images, text, and time series.

There are various activation functions used in neural networks, each with its own properties and advantages. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), Leaky ReLU, and softmax. These functions have different shapes and behaviors, allowing them to capture different types of non-linearities in the data.

Once trained, a neural network can generalize its learning to make predictions on new, unseen data. Neural networks have demonstrated remarkable success in various applications, including image and speech recognition, natural language processing, medical diagnosis, and autonomous driving, among others (28). They continue to be a key component of many cutting-edge artificial intelligence systems. Fig. 3.1 represents a NN.

3.2 Variational Quantum Circuits

Quantum circuits with variable parameters as gates can be employed to approximate a function to addresses a specific problem. The tunable parameters of the circuits are optimized iteratively by classical optimizers in the same way as NNs.

VQCs offer flexibility in terms of circuit depth and show some resilience to noise, which is crucial in the current scenario lacking robust quantum error correction. Despite limitations in current NISQ devices, quantum machine learning algorithms utilizing VQCs can navigate quantum errors effectively.

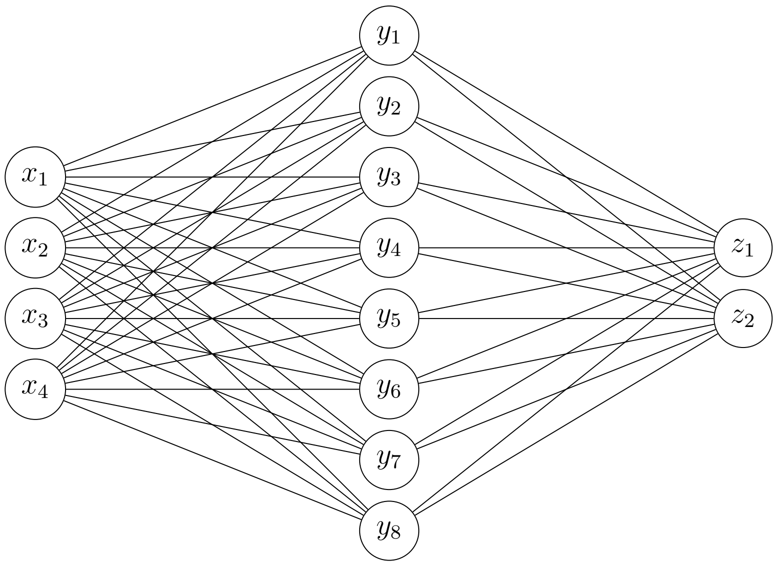


Figure 3.1: A simple 3 layered NN. Every connection represents a set of weights, biases and activation functions The whole neural network is a composition of all these functions.

Simulating quantum circuits with a large number of qubits using classical computers presents a formidable challenge due to the exponential growth in computational state-space dimensions. Although current quantum devices have limitations, recent breakthroughs, such as Google's demonstration of quantum sampling, highlight the potential of quantum computing.

VQCs, in some cases, require fewer parameters than conventional neural networks, making them promising for modeling complex environment (11). For example, simulating a 100-qubit physical system on classical computers demands immense computational resources. In contrast, a VQC could represent such systems with fewer parameters, potentially outperforming classical neural networks in terms of efficiency and scalability. Fig. 3.2 represents a VQC.

3.3 Q-Learning

RL stands as a foundational pillar in the field of artificial intelligence, aiming to enable agents to learn optimal behaviors through interaction with their environment. Among the various algorithms in reinforcement learning, Q-learning has emerged as a fundamental and widely utilized technique which

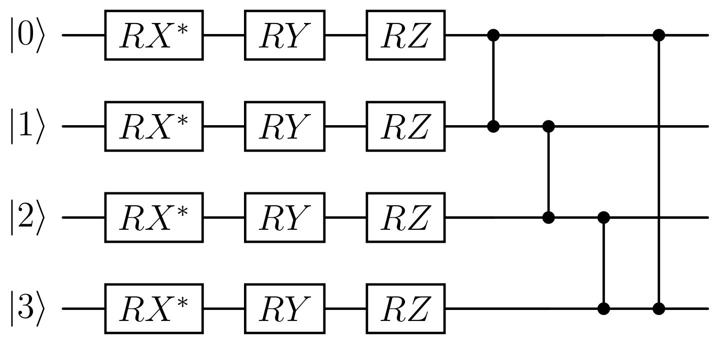


Figure 3.2: i^{th} Quantum layer ansatz. Gates with $(*)$ are non-trainable and used for encoding and data re-upload. RX, RY and RZ are rotational gates. Finally, the CZ gates are used to entangle qubits with their adjacent qubits.

is effectiveness in solving a diverse range of problems.

Q-learning is a model-free RL algorithm, meaning it does not require prior knowledge of the environment's dynamics (29). Instead, it learns through trial and error, gradually improving its decisions based on feedback received from the environment. Q-learning belongs to the class of value-based methods, where the goal is to learn a value function that estimates the expected return or cumulative reward associated with taking a particular action in a given state.

The central concept in Q-learning is the Q-function, also known as the action-value function. The Q-function maps state-action pairs to their corresponding expected returns, indicating the quality or utility of taking a specific action in a particular state. Mathematically, the Q-function is defined as follows:

$$Q(s,a) = E[R_t|s,a]$$

Where $Q(s,a)$ represents the expected return (cumulative reward) obtained by taking action a in state s, and $E[R_t|s,a]$ denotes the expected value of the sum of rewards starting from state s, taking action a, and following the optimal policy thereafter.

The key objective in Q-learning is to iteratively update the Q-values based on observed transitions in the environment. This is typically achieved using the Bellman equation, which expresses the relationship between the Q-values of successive states:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Where α is the learning rate, R is the immediate reward received after taking action a in state s, γ is the discount factor that controls the importance of future rewards, s' is the next state, and a' is the action chosen in the next state according to the current policy.

One of the key advantages of Q-learning is its ability to handle environments with large state and action spaces. This is made possible through the use of function approximation techniques, such as neural networks, to approximate the Q-function. Deep Q-learning, which employs deep neural networks to approximate the Q-values, has gained significant attention and achieved remarkable success in solving complex tasks, such as playing video games and controlling robotic systems.

Despite its effectiveness, Q-learning is not without its challenges and limitations. One notable issue is the trade-off between exploration and exploitation, where the agent must balance between trying out new actions to discover potentially better strategies (exploration) and exploiting its current knowledge to maximize rewards (exploitation). Various exploration strategies, such as epsilon-greedy and softmax exploration, have been proposed to address this challenge.

In recent years, advancements in Q-learning have extended its applicability to diverse domains such as robotics, autonomous vehicles, finance, and healthcare. Moreover, research efforts continue to explore novel variants and enhancements to Q-learning, such as double Q-learning, prioritized experience replay, and distributional Q-learning, to further improve its efficiency and stability.

3.4 Deep Q-Network

Deep Q-Network (DQN) leverages the power of deep neural networks to approximate the Q-function in complex environments. Building upon the foundation of Q-learning, DQNs overcome the limitations of traditional tabular Q-learning by enabling the handling of high-dimensional state spaces and complex decision-making tasks. DQNs extend the principles of Q-learning by employing deep neural networks, to approximate the Q-values.

The architecture of a DQN typically consists of multiple layers of neurons, with the input layer receiving state observations and the output layer representing Q-values for each action. During training, the network is trained to minimize the discrepancy between predicted Q-values and target Q-values, obtained using some variant of the Bellman equation.

The training process involves iteratively updating the network's weights to minimize a loss function, such as the mean squared error between predicted and target Q-values. This process is often facilitated by optimization algorithms like stochastic gradient descent (SGD) or its variants, which adjust the network's parameters to improve its performance over time.

Here's a step-by-step explanation of how the DQN algorithm works:

1. **Initialization**:
   - Initialize the DQN with random weights.
   - Define a replay memory buffer to store experiences (state, action, reward, next state, and done flags) encountered during training.
2. **Interaction with the Environment**:
   - Start interacting with the environment by selecting actions based on an exploration strategy (e.g., epsilon-greedy).
   - Execute selected actions in the environment and observe the resulting next state and reward.
   - Store the transition (state, action, reward, next state, done flag) in the replay memory buffer.
3. **Q-Network Update**:
   - Sample a minibatch of experiences from the replay memory buffer.
   - Compute the target Q-values for each experience using a variant of the Bellman equation.
   - Update the weights of the Q-network to minimize the temporal difference (TD) error between predicted and target Q-values.
   - This is typically done using gradient descent optimization algorithms like stochastic gradient descent (SGD) or its variants.
   - The loss function used for training the Q-network is typically the mean squared error between predicted and target Q-values.
4. **Target Network Update**:
   - Periodically update a target Q-network with the weights of the main Q-network.
   - This helps stabilize training by reducing the correlation between target and predicted Q-values.
   - The target Q-network is used exclusively for computing target Q-values and is not updated during the training process.

5. **Repeat**:
    - Continue interacting with the environment, updating the Q-network, and periodically updating the target network until convergence criteria are met.

Key Components of DQN:

- **Replay Memory**: Stores past experiences to break correlations between consecutive updates and improve sample efficiency.
- **Target Network**: A separate network used for computing target Q-values, updated periodically to stabilize training.
- **Exploration Strategy**: Determines how actions are selected during training, balancing exploration of new actions with exploitation of learned knowledge.
- **Q-Network Architecture**: Typically consists of deep neural network layers, such as convolutional and fully connected layers, to approximate the Q-function.

One of the key advantages of Deep Q-Learning is its ability to generalize across similar states, thereby improving sample efficiency and enabling effective learning from limited data. This is achieved through the use of function approximation techniques provided by deep neural networks, which capture underlying patterns and relationships in the data.

However, Deep Q-Learning also faces several challenges and considerations. Training deep neural networks can be computationally intensive and may require substantial computational resources, especially for complex environments with high-dimensional state spaces. Moreover, the stability and convergence of Deep Q-Learning algorithms can be affected by issues such as overestimation bias, non-stationarity of the target network, and exploration-exploitation trade-offs.

Despite these challenges, Deep Q-Learning has demonstrated remarkable success in various domains, including playing Atari games, controlling robotic systems, and optimizing complex decision-making processes. Ongoing research continues to explore enhancements and extensions to Deep Q-Learning, such as Double Deep Q-Learning, Dueling Q-Networks, and Prioritized Experience Replay, aiming to address its limitations and further improve its performance and applicability.

3.5 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a state-of-the-art RL algorithm that aims to efficiently optimize policy functions in tasks (30). Developed by OpenAI, PPO builds upon the advantages of policy gradient methods while addressing their limitations, such as high variance and sensitivity to hyperparameters. PPO achieves a balance between stability, sample efficiency, and scalability, making it widely adopted in various domains.

Traditional policy gradient methods, such as REINFORCE and Trust Region Policy Optimization (TRPO) (31), suffer from high variance and sensitivity to hyperparameters. These methods often require careful tuning of learning rates and clipping parameters to achieve stable training. Additionally, TRPO's computational complexity and memory requirements can be prohibitive for large-scale applications. PPO addresses these challenges by introducing a simple yet effective optimization objective that ensures stable and efficient learning.

PPO is based on the principle of trust region optimization, where policy updates are constrained to a region around the current policy to prevent large policy changes that may lead to instability. The key concepts of PPO include:

- **Clipped Surrogate Objective**: PPO introduces a clipped surrogate objective that constrains policy updates to a small region around the current policy. By limiting the magnitude of policy changes, PPO ensures stable and incremental updates.

- **Proximal Policy Optimization**: The clipped surrogate objective is optimized using proximal optimization techniques, such as the trust region constraint or penalty terms. This ensures that policy updates are conservative and do not deviate too far from the current policy.

- **Adaptive Learning Rates**: Adaptive learning rates are used to dynamically adjust the step sizes during optimization. This allows PPO to effectively navigate the optimization landscape and converge to a stable policy faster.

The training procedure of PPO involves the following steps:

1. **Collect Data**: Collect trajectories by interacting with the environment using the current policy.

2. **Compute Surrogate Objective**: Compute the clipped surrogate objective using the collected data and the current policy. Typically a parameter describes the extent of clipping.

3. **Optimize Objective**: Optimize the clipped surrogate objective using gradient-based optimization methods, such as stochastic gradient descent (SGD) or Adam.

4. **Update Policy**: Update the policy parameters using the optimized objective, ensuring that policy updates are within a trust region around the current policy.

5. **Repeat**: Repeat the process iteratively until the policy converges to an optimal solution or a stopping criterion is met.

PPO offers several advantages over traditional policy gradient methods:

- **Stability**: Stable training is provided by constraining policy updates to a small region around the current policy.

- **Sample Efficiency**: Efficient learning is achieved by leveraging adaptive learning rates and clipping objectives.

- **Scalability**: PPO is scalable to large-scale RL tasks, because of its low memory requirements and efficient optimization procedure.

PPO has been successfully applied to diverse RL tasks and its stability, sample efficiency and scalability make it a versatile tool.

3.6 Cartpole Environment

The CartPole environment has a 4 dimensional input space and represents a fundamental RL setting where the objective is to balance a pole atop a moving cart. With a 4-dimensional state space providing observations of the cart's position and velocity along with the pole's angle and angular velocity, agents make discrete movements, either left or right, to stabilize the pole (9).

Episodes terminate when the pole exceeds a critical angle or the cart reaches an edge or if the agent has completed a certain number of steps set by the user. Agents receive a reward for each time step the pole remains balanced, encouraging them to maximize cumulative rewards. This environment serves as a foundational and widely-used benchmark in reinforcement learning, offering a simple yet challenging setup for testing and developing various learning algorithms and control strategies.

Environment Description

- **Physical Setup**: The CartPole environment consists of a cart that moves along a frictionless track and a pole attached to the cart by a joint.
- **Observation Space**: The state of the environment is represented by a vector of four real numbers: cart position, cart velocity, pole angle, and pole angular velocity.
- **Action Space**: The agent can take two discrete actions: push the cart to the left or push the cart to the right.
- **Reward Structure**: The agent receives a reward of $+1$ for each time step that the pole remains upright. The episode terminates if the pole falls beyond a certain angle threshold or if the cart moves outside a predefined boundary.

Reinforcement Learning Setup

- **Episode Termination**: Episodes terminate when the pole falls beyond a certain angle threshold or when the cart moves outside a predefined boundary. Upon termination, the episode resets to its initial state.
- **Training Objective**: The agent aims to learn a policy that maximizes the cumulative reward obtained over a sequence of actions. Reinforcement learning algorithms, such as Q-learning or policy gradient methods, are used for training.
- **Evaluation Metrics**: Performance is evaluated based on metrics such as the average duration of time steps that the pole remains upright, average reward per episode, or convergence speed of the learning algorithm.

3.6.1 Noise

Three types of noises are simulated in the CartPole game.

- **Gaussian Noise**: Common in sensory data, it adds zero-mean Gaussian noise to observed states, with variance aligning to the variance across all recorded states.

- **Observation Blackout**: Results in the complete loss of observational information by setting interfered states to zero.
- **Frozen Observation**: Indicates delays and frozen observations due to limited data communication, resulting in lagged observations. The state becomes the previous state if the perturbation remains constant.

The noise is added randomly to the input state with a set probability when the agent is playing the game during testing and training.

Chapter 4

Experiments

4.1 Networks

All the classical layers are created using PyTorch (32) and the quantum circuits are created using pytorch-quantum (33). Classical neural networks within the model consist of three fully connected layers with ReLU activation after every layer, except for the final output layers (Q-Values and interference probability). The quantum and hybrid networks make use of VQCs and hybrid VQCs in place of purely classical networks. The network ansatz used across all the VQCs are the same as in and are shown in Fig. 3.2. The hybrid networks make use of classical preprocessing layers before encoding to the VQC as shown in Fig. 4.1 Data is encoded to the qubits by applying an RX gate with the input values as the parameter. A single layer consists of consecutive trainable variational gates on all the wires - RY, then RX, and finally CZ gate is applied across adjacent qubits to entangle them. Measurements of the circuit are of two types - single qubit and multi-qubit correlation. Single qubit measurements return the expected value of the Pauli Z operation on each qubit. Multi-qubit measurements return the correlation between the expected value of the Pauli Z operation on multiple qubits. These can be used to change the size of the output vector. The circuits make use of data re-uploading - before each layer, the original input is encoded using the RX operation on the qubits.

4.2 DQN Algorithm

In one single step of the algorithm, the Cartpole environment is first refreshed to provide the current state. We then choose to attack the state with a noise with a certain attacking probability. Then the state is used as

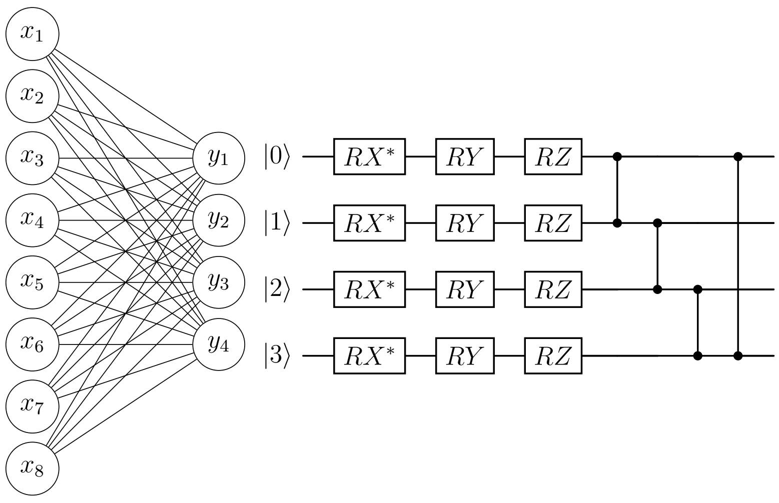


Figure 4.1: Sample hybrid classical and quantum layer. Using hybrid models in this way can be useful to reduce the number of qubits used in the quantum circuit. A large input can first be processed by the classical network to process it into a smaller sized input for the quantum circuit.

input for the agent which employs the local network to produce action values and interference probabilities as outputs. The agent may choose to use these action values to play the game, or choose a random action to explore the environment according to an exploration-exploitation policy. Here, we use an exponentially decaying threshold to choose between exploration and exploitation.

The chosen action from the agent is sent to the environment, which returns a reward, the next state and information about the state. We store the current state, action, reward, next state, whether the episode ended or not, and if the state was interfered or not to the replay buffer. These will be used for training the network later.

Once the step is over, the environment is refreshed again and this is repeated until a certain termination step is reached, or the agent loses. Then the next episode is played. This process is repeated for 50,000 steps. The agent has a training step between every 20 steps played. The target network is updated with the parameters of the local network after the training step.

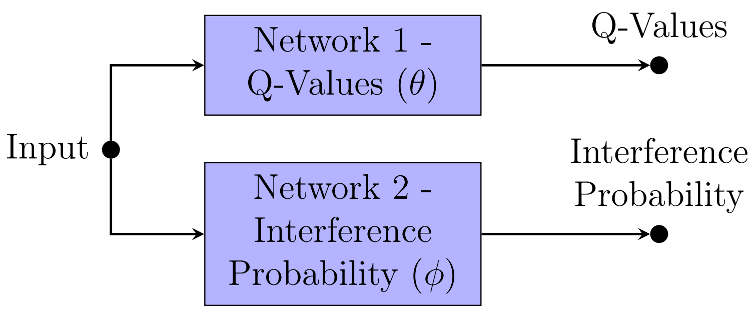

Network 1 - Q-Values (θ)
Q-Values
Input
Network 2 - Interference Probability (ϕ)
Interference Probability

Figure 4.2: Simple Architecture. θ and ϕ are the trainable parameters of each network. Note that in this architecture, the Interference Probability is not used for treatment selection, and is just trained separately to test whether the network is able to predict interference from the input.

4.2.1 Model architectures

The simple DQN and CIA DQN are implemented using classical, quantum and hybrid networks.

The simple DQN architecture consists of two networks. Network 1 takes the current state of the environment as input and processes it to produce actions. Network 2 also takes the state as input and produces logits to predict interference.

The CIA DQN, on the other hand, employs a causal inference models, utilizing binary treatment information and hidden confounders. This model maps interfered observations to a latent state, predicts the interference label, and adapts its policy based on this estimate.

The CIA DQN architecture as shown in Fig. 4.3 uses Network 1 to approximate a latent state from input. Then, Network 2 predicts the probability of interference of the latent state. Then, we choose between either Network 3 or Network 4 to produce Q-values depending on indication of Network 2 on interference. We use variable treatment for processing the latent state and playing the game, based on the interference.

The overall CIA model involves training the Q-network end-to-end using the DQN algorithm, including an additional loss for predicting the interference label. The CIA's objective function combines losses related to DQN training and interference label prediction. The approach aims to demonstrate the resilience of the CIA against observational interference, showcasing how the model can handle noisy observations.

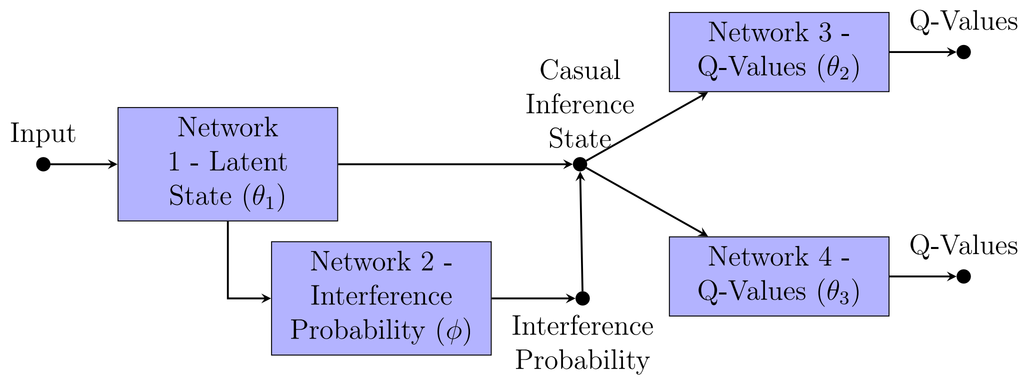

Input
Network 1 - Latent State (θ_1)
Network 2 - Interference Probability (ϕ)
Casual Inference State
Interference Probability
Network 3 - Q-Values (θ_2)
Network 4 - Q-Values (θ_3)
Q-Values
Q-Values

Figure 4.3: CIA Architecture. In this architecture, the Interference probabilities are used to weight the Q-Values from Network 3 and Network 4.

4.2.2 Agent

An agent class is created which makes use of the models for decision-making. The AdamW optimizer (34) is used to update the parameters of the networks while training. The learning rate used for all classical trainable parameters is 0.001 and for all quantum parameters is 0.01.

The agent has a replay buffer of size 10^5 which stores the experiences collected by the agent while playing instances of the game. The buffer stores state, next state, action, reward and interference label. Random batches of size 64 are sampled from the buffer for training the network.

During training, the local network is given the current state and the interference label and it outputs the Q-Values. Note that since the interference label is provided during training, this is used as a probability of 0 or 1 for interference. The 3rd and 4th network in the CIA will switch based on the provided label itself.

The target network similarly predicts the Q-Values for the next state. The loss is calculated between the two Q-Values of the networks. Since the networks have 2 outputs - Q-Values and interference probabilities, loss functions for the CIA and simple models are used as in (6):

$$\begin{aligned} L^{CIADQN}(\theta_1, \theta_2, \theta_3, \phi) = {} & i_t^{train} \cdot MSE(\theta_1, \theta_2, \phi) \\ & + (1 - i_t^{train}) \cdot MSE(\theta_1, \theta_3, \phi) \\ & + \lambda \cdot \left[i_t^{train} \log p(\tilde{i}_t | \tilde{z}_t; \theta_1, \phi) \right. \\ & \left. + (1 - i_t^{train}) \log\left(1 - p(\tilde{i}_t | \tilde{z}_t; \theta_1, \phi)\right) \right] \end{aligned} \tag{4.1}$$

$$
\begin{aligned}
L^{simpleDQN}(\theta_1,\theta_2,\phi) = & MSE(\theta_1,\theta_2) \\
& + \lambda \cdot \left[i_t^{train} \log p(\tilde{i_t}|\tilde{z}_t;\theta_1,\phi) \right. \\
& \left. + (1 - i_t^{train}) \log\left(1 - p(\tilde{i_t}|\tilde{z}_t;\theta_1,\phi)\right) \right]
\end{aligned} \tag{4.2}
$$

where θ_1, θ_2, θ_3 and ϕ are the trainable parameters of each network as shown in Fig. 4.2 and Fig. 4.3. This calculates the loss between the output Q-Values. i_t^{train} is the interference label that is provided to the network while training. λ is a scaling constant and is set to 1 for simplicity. p is the interference probability that is outputted from the network. $\tilde{i_t}$ is the interference probability predicted by the network and $\tilde{z}_t$ is the latent state at time t approximated by the network.

4.2.3 Training

All the models are trained using the same procedure. RL agents are susceptible to instabilities while training and converging. So for each model, the training procedure is repeated 10 times to obtain 10 trained networks. The learning curves of these models indicate the average training time and data sample efficiency of these algorithms.

The agent plays the CartPole game and collects experiences from the environment (tuple of state, action, reward, next state, done flags and interference label) in the replay buffer after every step. After every 20 steps, the agent trains its model to improve its strategies based on the data collected. We train the local network in 10 epochs. The agent samples 64 instances from the replay buffer. The target Q-values are calculated for the states and a loss is calculated with the predicted Q-values from the local network. The local network is then optimized using the AdamW optimizer. Once this process occurs for 10 epochs, the target network is updated with the weights of the local network.

After every 500 steps of training, the models are validated. This style of implementation allows us to test the learning of the models at standardized intervals, allowing us to accurately compare the training speeds and sample efficiency of different models. The validation step simple resets the environments, and allows to agent to play 10 episodes of the game with only exploitation. We take the average score to be the test or validation score at a particular time in the training procedure.

We define the stopping criteria for the training procedure to be when the average score of the past 10 validation scores reaches 195 or above. Once the training is complete, the models are ready to be tested for their performance.

The metric used to quantify the performance of models is explained in the next subsection.

4.2.4 Testing

Two performance metrics are employed to benchmark and compare the trained models. The first one compares how well the models fare in in different levels of noise called the performance score. It directly measures the ability of the models to play the game. The second metric tries to quantify how robust the models are, called the robust score.

For the performance score, the trained agents are made to play the Cart-Pole game, but with a cap of 500 steps per episode instead of 200. The agents do not explore or optimize anymore. The performance scores are tested for no noise, and varying ratio of attacks from frozen observation, Gaussian and blackout noise.

The performance of the agents are tested using the fully trained models. Every agent plays 50 games with a given interference type and ratio. The average score of these 50 games is taken as the performance score for each of the agents. The performance score assigned to the model architecture is taken as the best score from the 10 trained agents for each architecture. This quantity can be interpreted as how consistently a well-trained model can perform in the described scenarios.

For the robust score, we try to find how similar the agents react to original and noise-added inputs. We give the agents a batch of 100 original inputs, and then we add noise to the inputs and calculate the ratio for which the agents produced the same final actions. This metric quantifies the resilience of the models to various noises.

4.3 PPO Algorithm

This PPO implementation runs 16 parallel environments. These environments produce batches of 16 states as outputs and take batches of 16 actions as input for updating the states of the environments. Then the environment outputs 16 next states and 16 done flags - which is a binary variable indication whether an episode of the game has been completed or not. The environments may be implemented sequentially, or in parallel.

The steps in the algorithm consists of two phases - The rollout phase, and the learning phase.

Rollout phase: The agent takes the 16 states as input and gives 16 actions as outputs. The agent also stores the data of current state, action, reward, next state, done flags, and if the state was interfered or not to empty lists. This is however, different from the replay buffer used in the DQN algorithm, as we only store a current batch of data and discard it after the learning phase. This process is looped in all 16 environments for 128 steps.

Learning phase: During the learning phase, the agent utilizes the collected data from the rollout phase consisting of states, actions, rewards, next states, and done flags. This data is organized into batches of length 16 $\times$ 128 (number of environments $\times$ number of steps per environment).

The agent estimates values for the next observations which have been stored from the rollout, conditioned on whether the episode terminated (done flags). Then we calculate the advantage and return vectors for the optimizer step.

4.3.1 Model architectures

The simple PPO and CIA PPO are also implemented using classical, quantum and hybrid networks similar to the DQN. The major difference here is presented in method of implementing these networks. For the DQN algorithm, it is sufficient to obtain 2 action values as output. The PPO algorithm employs an actor model and a critic model. The actor produces action probabilities, and the critic is used to estimate the advantages. In this thesis, we replace the actor with the simple and CIA models to estimate values. For the critic, we simply employ a neural network with 3 layers and ReLU activation after the first two layers, similar to the other networks in the models and in Fig. 4.2 and Fig. 4.3

The simple PPO actor uses the same architecture as the simple DQN as in Fig. 4.2. The actor model produces two action values from Network 1 and two interference probabilities as logits from Network 2. The critic model uses only a single network which takes a state as input and produces a single value as output for every input state.

The CIA PPO actor uses the same architecture as the CIA DQN as in Fig. 4.3. Since the model's hypothesis is that a latent state has a strong causal relationship with the Q-values, the PPO model implemented also assumes that the critic advantage value depends on this latent state and not the original input state. Hence, The critic network takes the latent state produced by the actor as the input instead of the original state provided and produces a single critic value.

4.3.2 Agent

The optimizer settings used are the similar to the DQN. The AdamW optimizer (34) is used to update the parameters of the networks while training. The learning rate used for all classical trainable parameters is 0.001 and for all quantum parameters is 0.01. The rollout phase consists of 16 environments and 128 steps per environment. The clipping coefficient used is 0.2.

The overall loss function used is:

loss = policy loss − entropy × (entropy coefficient) + value loss × (value coefficient) + interference loss

$$\begin{aligned} L^{\text{CLIP+VF+S+I}}(\theta) = \hat{\mathbb{E}}_t \big[& L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \big] \\ & + \lambda \cdot \big[i_t^{train} \log p(\tilde{i_t} | \tilde{z}_t; \theta_1, \phi) \\ & + (1 - i_t^{train}) \log (1 - p(\tilde{i_t} | \tilde{z}_t; \theta_1, \phi)) \big] \end{aligned} \tag{4.3}$$

where θ is the set of all parameters of the model. i_t^{train} is the interference label that is provided to the network while training. λ is a scaling constant and is set to 1 for simplicity. p is the interference probability that is outputted from the network. $\tilde{i_t}$ is the interference probability predicted by the network and $\tilde{z}_t$ is the latent state at time t approximated by the network.

4.3.3 Training

The training occurs in 8 epochs and 8 minibatches during the learning phase. The optimizer updates the weights of the networks based on the PPO loss. In the PPO as well, for each model, the training procedure is repeated 10 times to obtain 10 trained networks. The learning curves of these networks indicate the average training time and data sample efficiency of these algorithms.

When the 100-episode average score reaches 195 or above, the agent is considered to be trained. Once the training is complete, the models are ready to be tested for their performance.

4.3.4 Testing

The PPO employs the same metrics to test performance and robustness and the DQN model. The trained actor of the PPO functions exactly like the trained local network of the DQN. These networks are ready to be employed in a testing scenario to play in the environment.

Chapter 5

Results

5.1 DQN

5.1.1 Sample Efficiency and Learning Speed

To get an accurate idea of sample efficiency of the models, we need a uniform method to compare across different training runs. So we validate the model at specific steps (after every 500 steps) to check the progress of agent's learning. Here, the model does not learn, but simply plays the game 10 times until termination and takes the average score. We plot these validation scores alongside the number of steps of the algorithm, which now provides a standard x-axis that is comparable across runs.

The termination for DQN is set at 200 as the models were unable to complete training till 500 steps in the given set up. We notice from Fig. 5.1 that both the classical simple and CIA models with low parameters is underperforming compared to the rest. The quantum and hybrid models with much lower number of parameters are able to training with much higher sample efficiency. However, classical networks have the advantage of being able to scale up the number of parameters easily. The classical models with high parameters are able to learn and complete the entire training within 30000 steps of the algorithm.

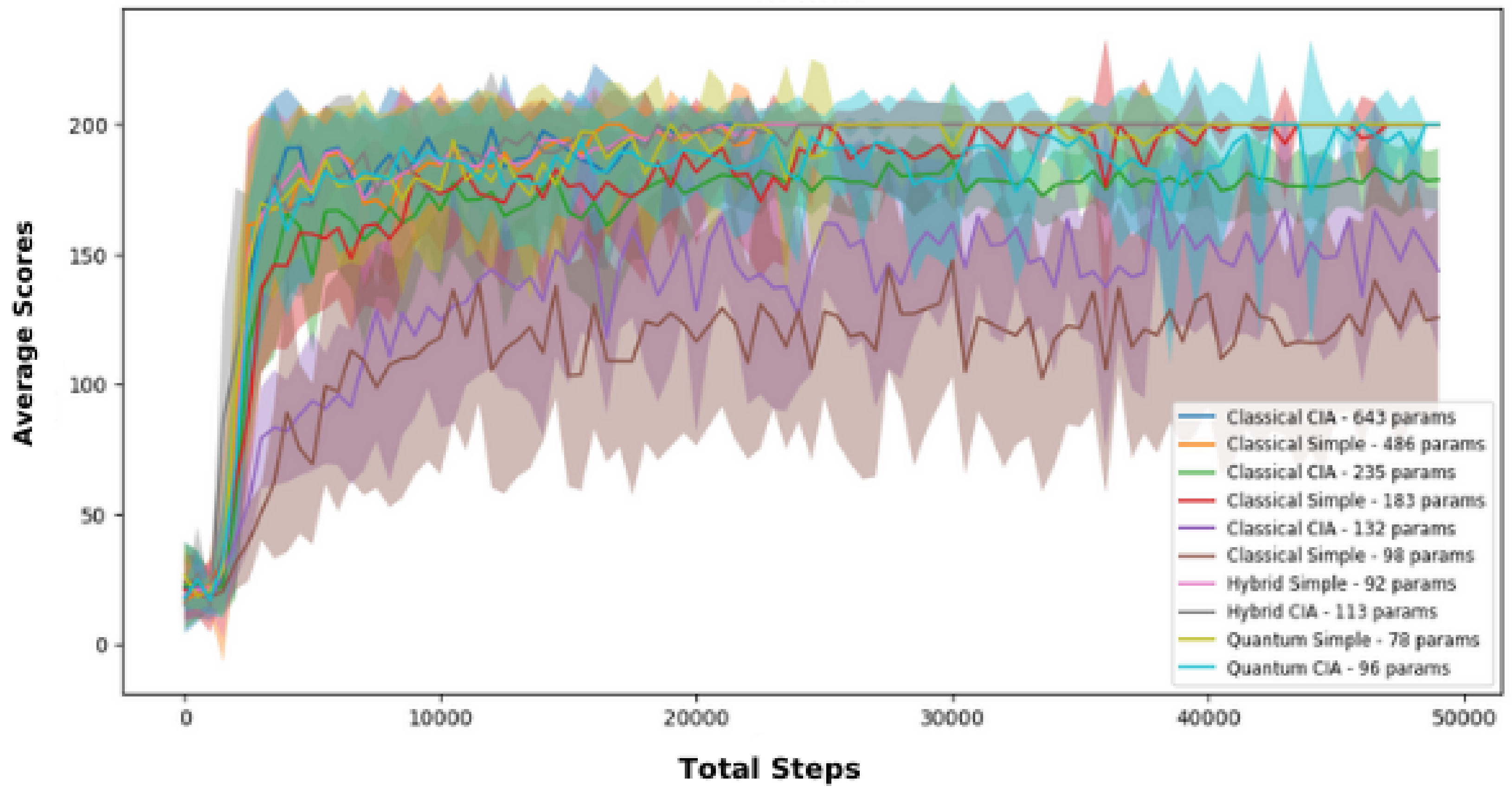


Figure 5.1: **Average validation scores of models in DQN - No Noise** Only the Quantum simple model and the two classical models with low parameters are not able to learn and converge.

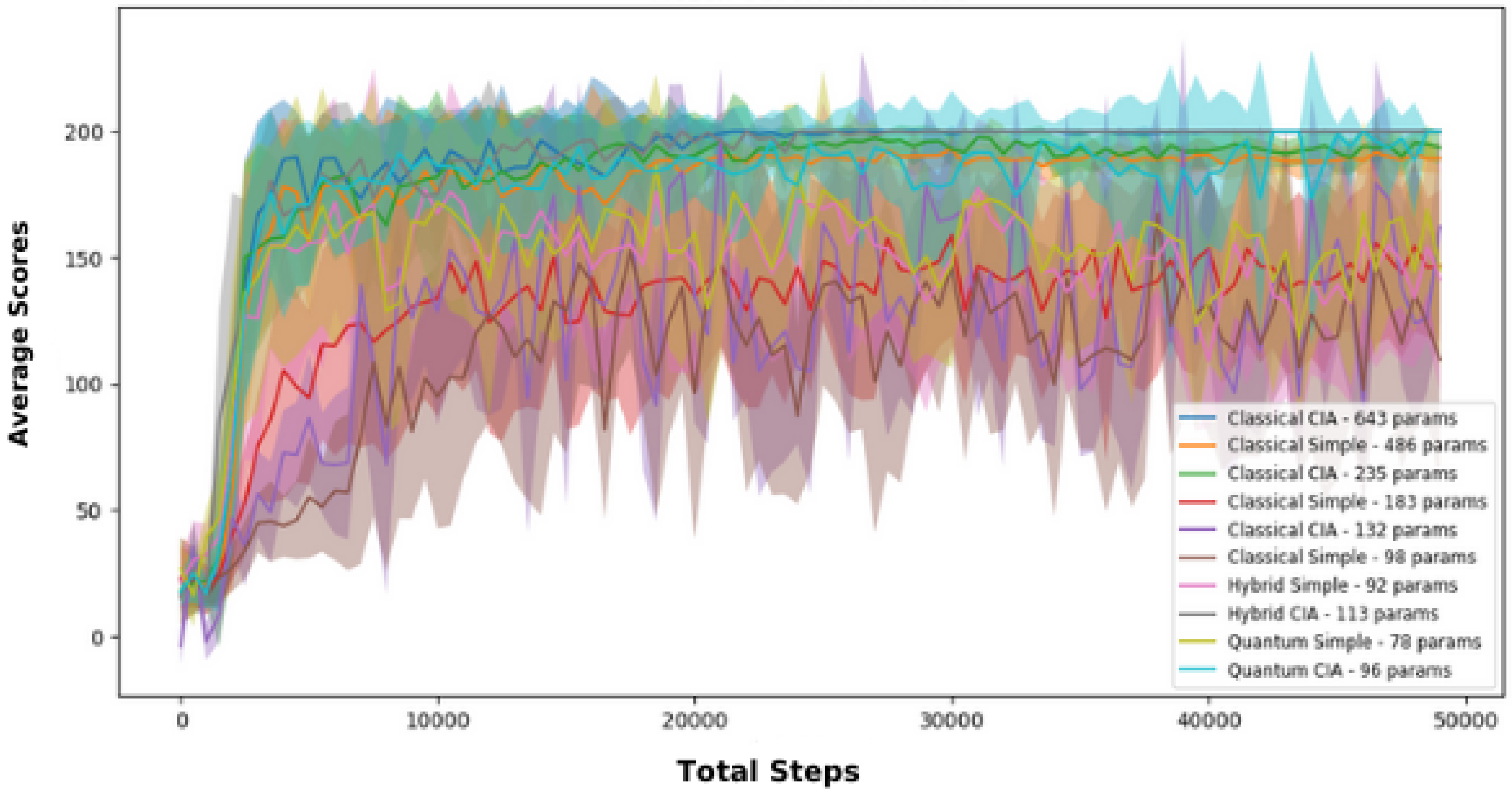


Figure 5.2: **Average validation scores of models in DQN - Frozen Observation Noise** The CIAs perform almost as well as the case with no noise. The simple models are learning much slower than before, or stagnate around a score.

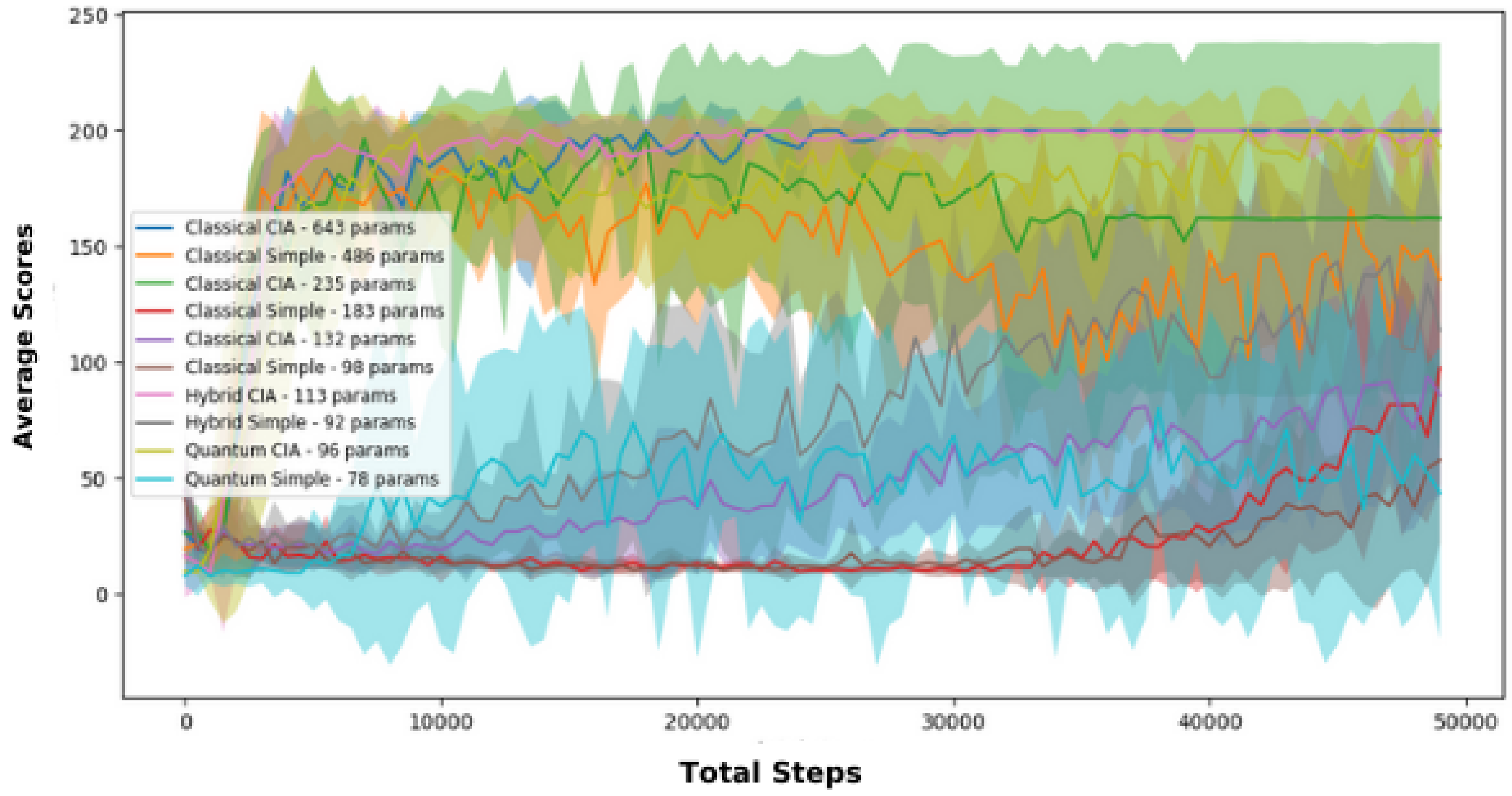


Figure 5.3: **Average validation scores of models in DQN - Blackout Noise** There seems to be a very noticeable difference between the resilient and the under-performing models under Blackout noise.

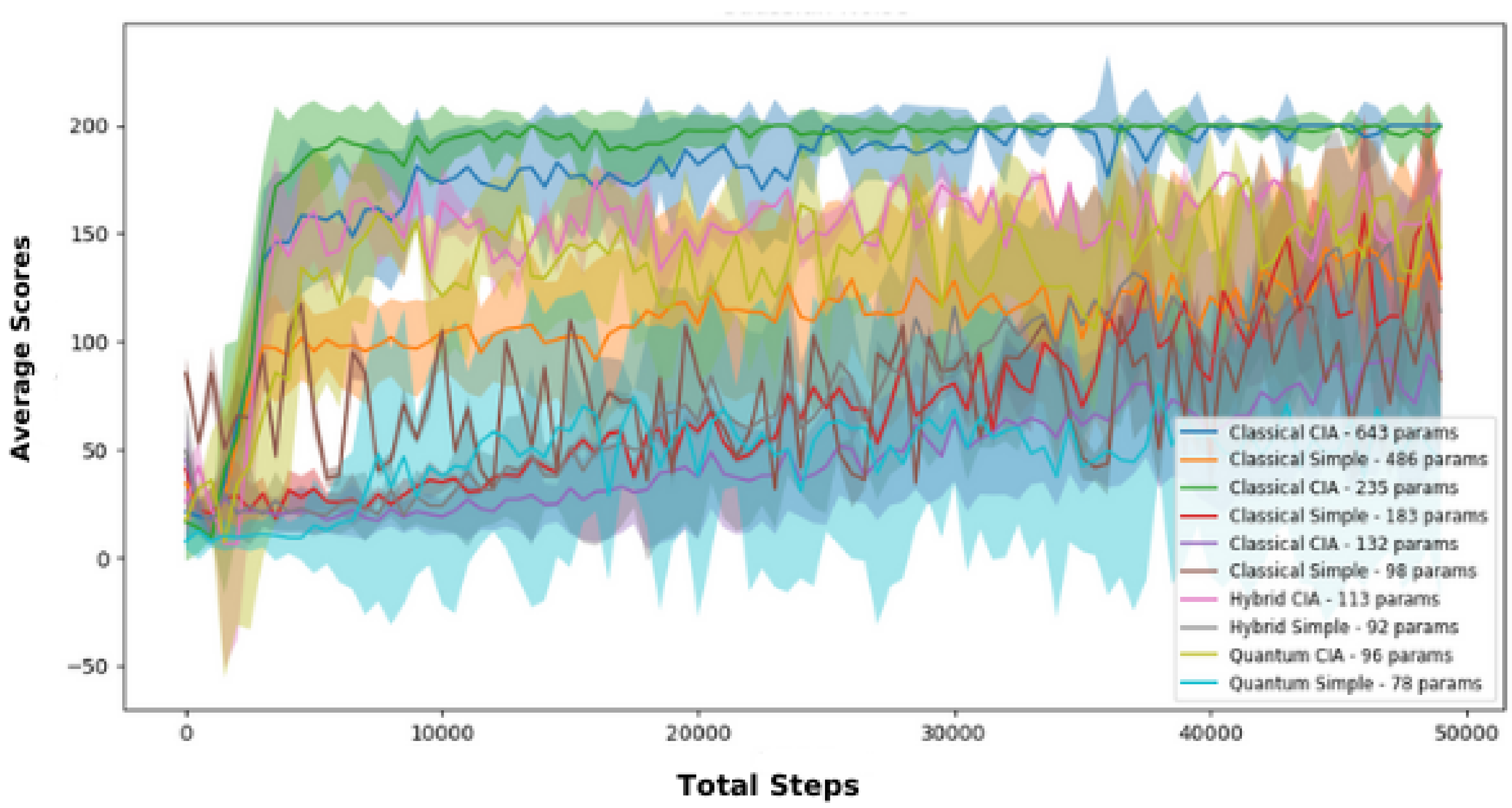


Figure 5.4: **Average validation scores of models in DQN - Gaussian Noise** The variance of the simple models seem to be highest under Gaussian noise.

Table 5.1: Performance Test Scores of the DQN models tested with no noise

Network Type	Number of Parameters	Avg. Score (Out of 500)
Classical Simple	98	189.56
Classical Simple	183	263.08
Classical Simple	486	467.03
Classical CIA	132	289.18
Classical CIA	235	407.27
Classical CIA	643	465.31
Hybrid Simple	92	342.16
Hybrid CIA	113	457.2
Quantum Simple	78	374.06
Quantum CIA	96	472.14

We notice similar patterns in Fig. 5.2, 5.3 and 5.4 with noise as well. We see that the quantum and hybrid models are competitive with classical models with about three times the number of parameters. When noise is introduced, we see a clear difference between the CIA models and the simple models. We see that the CIA models are able to efficiently learn while using lesser data.

Blackout noise seems to have the highest effect in regards to separating the learning speeds of the simple and CIA models. The frozen observation noise seems to affect this trend the least. This could be because the Cartpole game is simple enough that the models are able to learn to navigate the environment well even using just the previous frame.

5.1.2 Performance Scores

Each model is trained on frozen observation, Gaussian noise and blackout interference. Every model is then tested on the same interference type as they were trained on, varying the ratio of interference from 0%, 10%, 20%, 30%. It is to be noted that performance scores have significant variance. However, they qualitatively indicate the trends in performance of the networks.

From Table 5.1, we can infer that using quantum layers in hybrid models and fully quantum models can help in reducing the total number of parameters used in the model without degrading performance. We notice that the Quantum CIA model has the highest score with only 96 parameters. It performs at par with the largest network trained - the Classical CIA model that uses 643 parameters.

In all 4 cases, the simple models seem to follow a trend of lowering performance as the interference ratio increases. Such a stark trend is not noticeable

Table 5.2: Performance test scores of the DQN models tested with Frozen Observation noise

Network Type	**Number of Parameters**	**Avg. Score (Out of 500) for Interference Ratio**		
		10%	**20%**	**30%**
Classical Simple	98	132.8	111.24	70.22
Classical Simple	183	225.8	198.72	76.3
Classical Simple	486	470.88	214.0	76.34
Classical CIA	132	264.52	302.74	276.34
Classical CIA	235	326.16	294.2	344.6
Classical CIA	643	412.6	458.83	397.93
Hybrid Simple	92	128.4	142.45	71.8
Hybrid CIA	113	256.8	387.8	424.64
Quantum Simple	78	193.1	137.6	93.6
Quantum CIA	96	436.7	342.53	328.9

in the CIAs which shows the potential improvement in resilience of this model compared to the simple models. We also note that the simple models are able to perform competitively when the interference ratio is low from Table 5.2, 5.3 and 5.4. In Table 5.3, the Classical simple model achieved a perfect 500 score with 486 parameters and 10% Gaussian noise. However, their performance degrades quickly as the interference ratio increases. In Table 5.2, we see that at 30% interference, all the simple models score less than 100, while the CIAs are able to still play competitively.

These trends show promise for the CIA in DRL applications. We also see the advantage of using quantum networks, either standalone or as hybrid, as they are able to model the same function to play the game with with much lesser parameters. This could mean that if the quantum networks are scaled up, they could perform better than the classical networks in DRL situations.

5.1.3 Robust Scores

The robust scores is a measure of how resilient the models are to noise. We provide the trained models with 1000 inputs of noisy observations and their corresponding observation without noise. We compare the number of predictions where the model outputs the same action for both of these cases.

We notice a similar trend from Table 5.5 that classical models with higher parameters have higher resilience. It can also be noted that the CIA helps improve the resilience of these networks. This is true across all implementations - classical, quantum and hybrid. However, this does not directly indicate that these models will perform better in the game scenario, but simply indicates

Table 5.3: Performance test scores of the DQN models tested with Gaussian noise

Network Type	**Number of Parameters**	**Avg. Score (Out of 500) for Interference Ratio of**		
		10%	**20%**	**30%**
Classical Simple	98	179.9	177.28	176.28
Classical Simple	183	243.4	240.9	248.12
Classical Simple	486	500	486.08	427.86
Classical CIA	132	318.9	242.14	164.93
Classical CIA	235	374.14	294.2	336.0
Classical CIA	643	464.12	438.6	375.6
Hybrid Simple	92	241.5	274.74	227.56
Hybrid CIA	113	321.8	306.5	294.32
Quantum Simple	78	279.8	147.8	96.8
Quantum CIA	96	465.7	392.52	374.9

the robustness.

5.2 PPO

5.2.1 Sample Efficiency and Learning Speed

In the case of PPO, its more intricate to create a standard validation score. The training scores are measured from the parallel environments in which the PPO is playing, and it measures the average scores after each episode in the training phase from all the parallel environments and assigns a score at a standard time step.

The termination for PPO is set at 500 as the models were able to complete training till 500 steps quickly in the given set up. We notice from Fig. 5.5 that both the classical simple and CIA models with low parameters is under-performing compared to the rest, similarly to the DQN. The quantum and hybrid models with much lower number of parameters are able to training with much higher sample efficiency.

We notice similar patterns in Fig. 5.6, 5.7 and 5.8 with noise as well. We see that the quantum and hybrid models are competitive with classical models with about three times the number of parameters. When noise is introduced, we see a clear difference between the CIA models and the simple models. We see that the CIA models are able to efficiently learn while using lesser data.

Blackout noise and gaussian noise both seem to have high effects in re-

Table 5.4: Performance test scores of the DQN models tested with Blackout noise

Network Type	**Number of Parameters**	**Avg. Score (Out of 500) for Interference Ratio of**		
		10%	**20%**	**30%**
Classical Simple	98	153.66	140.72	77.06
Classical Simple	183	217.88	194.0	173.36
Classical Simple	486	468.12	309.3	165.52
Classical CIA	132	362.5	277.83	325.75
Classical CIA	235	385.6	319.4	357.76
Classical CIA	643	485.6	454.66	330.4
Hybrid Simple	92	203.22	197.52	132.58
Hybrid CIA	113	316.6	348.0	297.38
Quantum Simple	78	186.47	212.9	81.47
Quantum CIA	96	364.5	338.7	358.85

Table 5.5: Robust scores of the DQN models

Network Type	**Number of Parameters**	**Robust Score for Noise Type of**		
		Frozen Observation	**Gaussian**	**Blackout**
Classical Simple	98	65.4%	54.7%	71.2%
Classical Simple	183	77.8%	84.4%	68.9%
Classical Simple	486	88%	79.3%	72.3%
Classical CIA	132	78.8%	82.9%	91.3%
Classical CIA	235	92.3%	88.4%	84.5%
Classical CIA	643	98.7%	96%	92.5%
Hybrid Simple	92	85.7%	77.3%	75.8%
Hybrid CIA	113	95.6%	87.8%	91.1%
Quantum Simple	78	73.2%	66.7%	78.4%
Quantum CIA	96	92.2%	83.4%	89.2%

gards to separating the learning speeds of the simple and CIA models. The frozen observation noise seems to affect this trend the least.

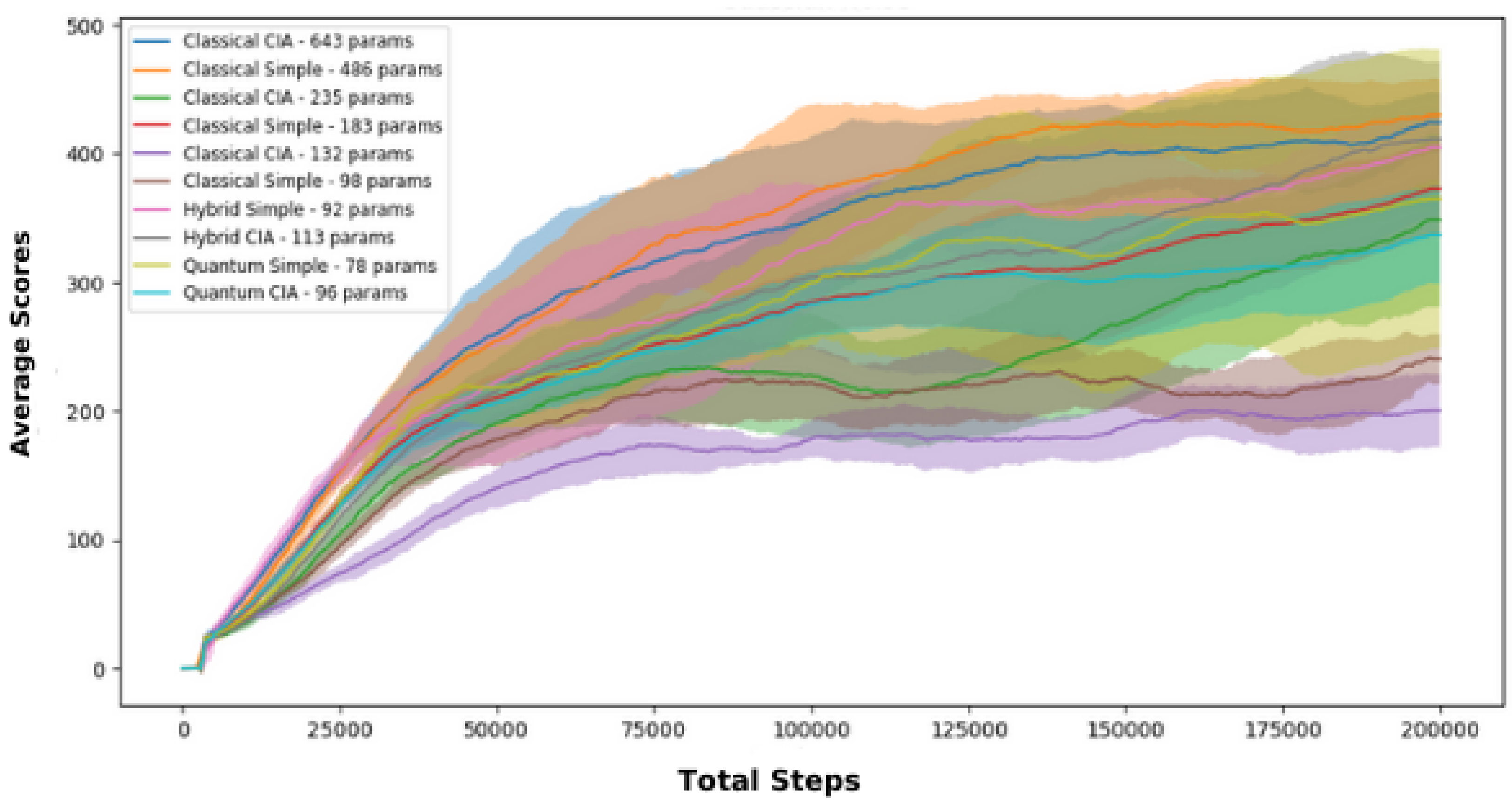


Figure 5.5: **Training steps vs. episode scores of PPO - No Noise** The training scores of the PPO have been interpolated such that the number of episodes can be mapped to the number of steps to ensure standardization of sample efficiency. This graph shows the mean of the moving average training scores from the 10 networks.

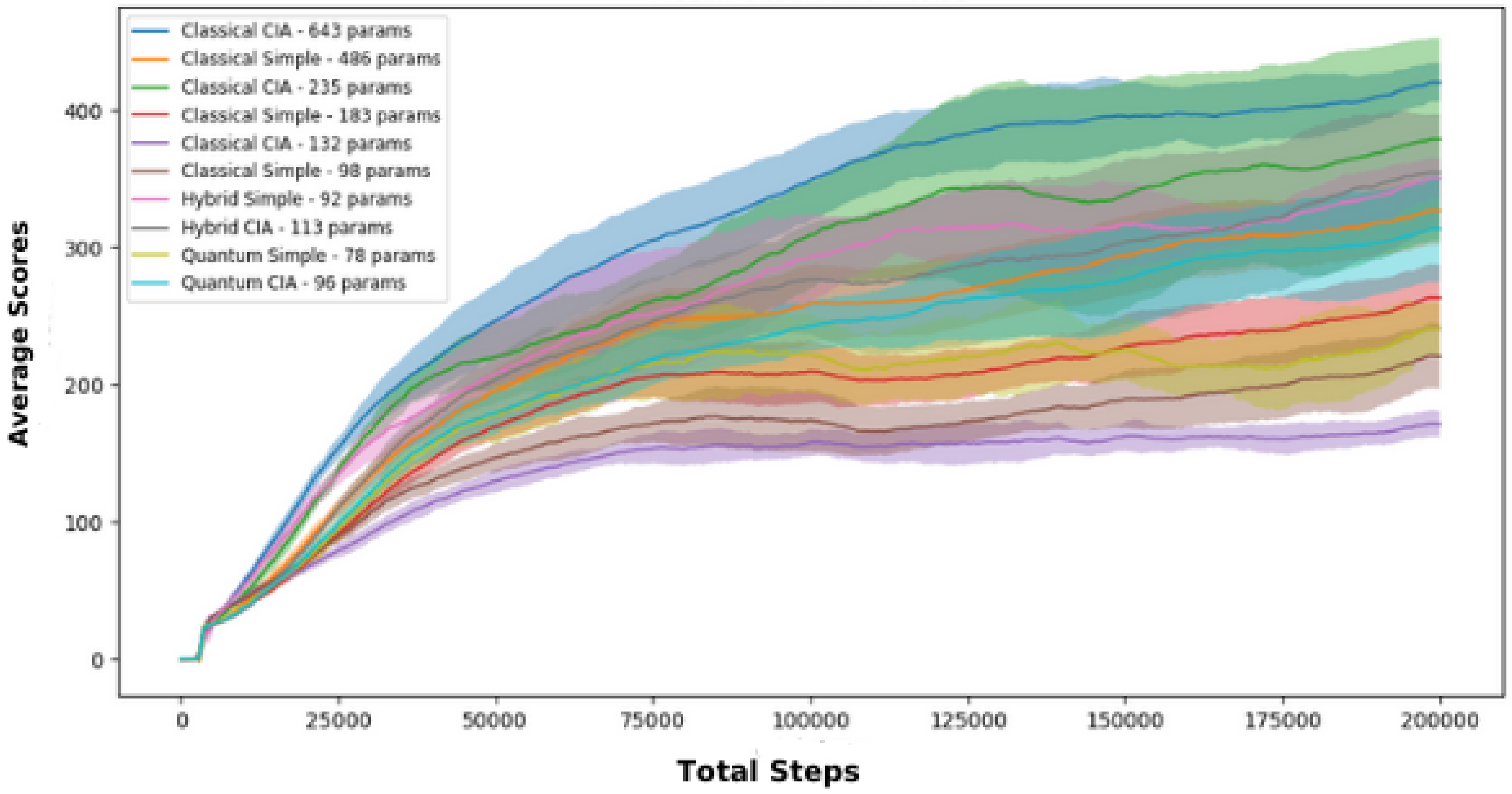


Figure 5.6: **Training steps vs. episode scores of PPO - Frozen Observation noise** We are able to notice clear striations in the learning speeds of the different networks.

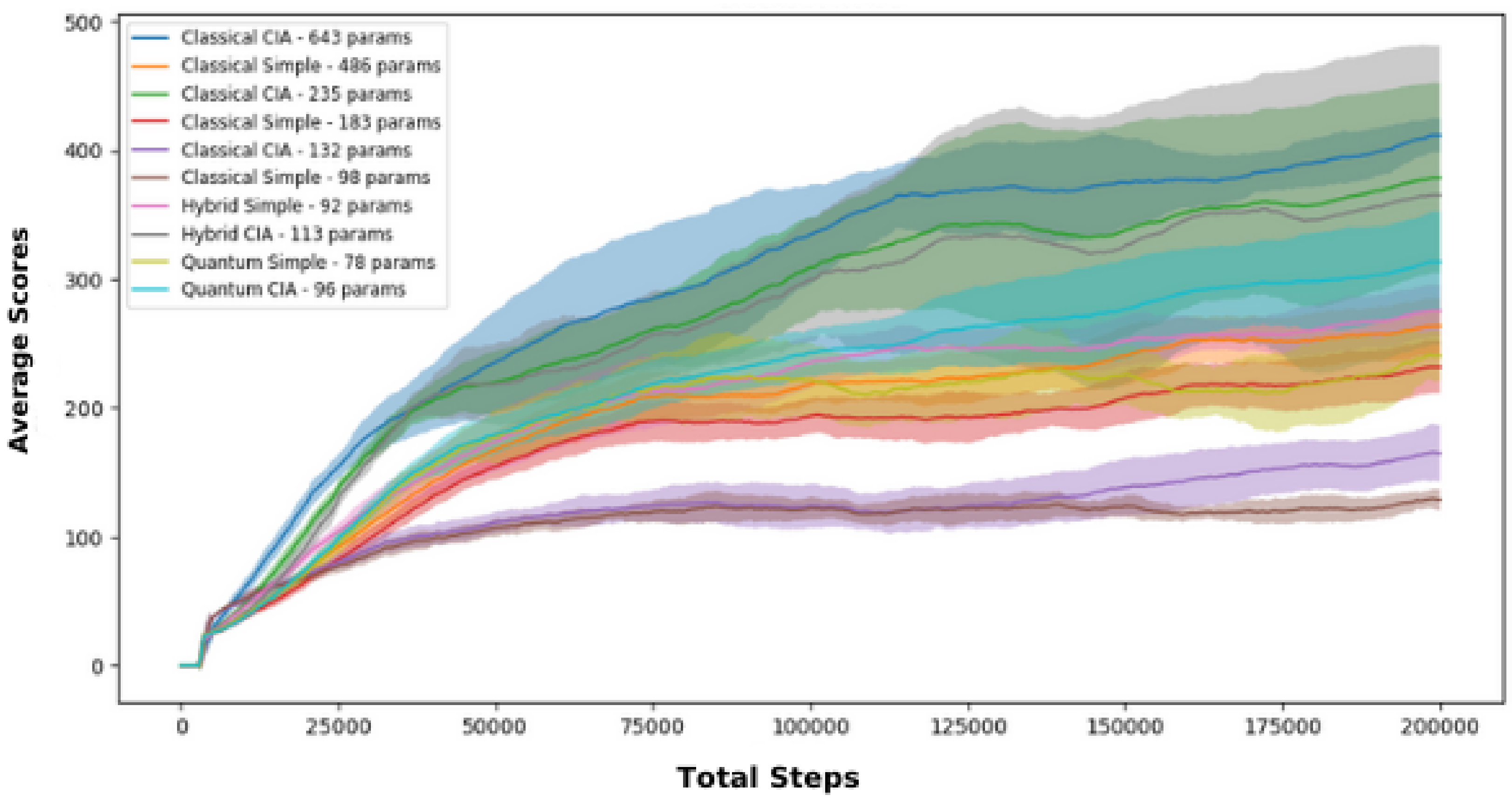


Figure 5.7: **Training steps vs. episode scores of PPO - Blackout Noise** Blackout noise seems to differentiate the sample efficiency of the models to the highest extent.

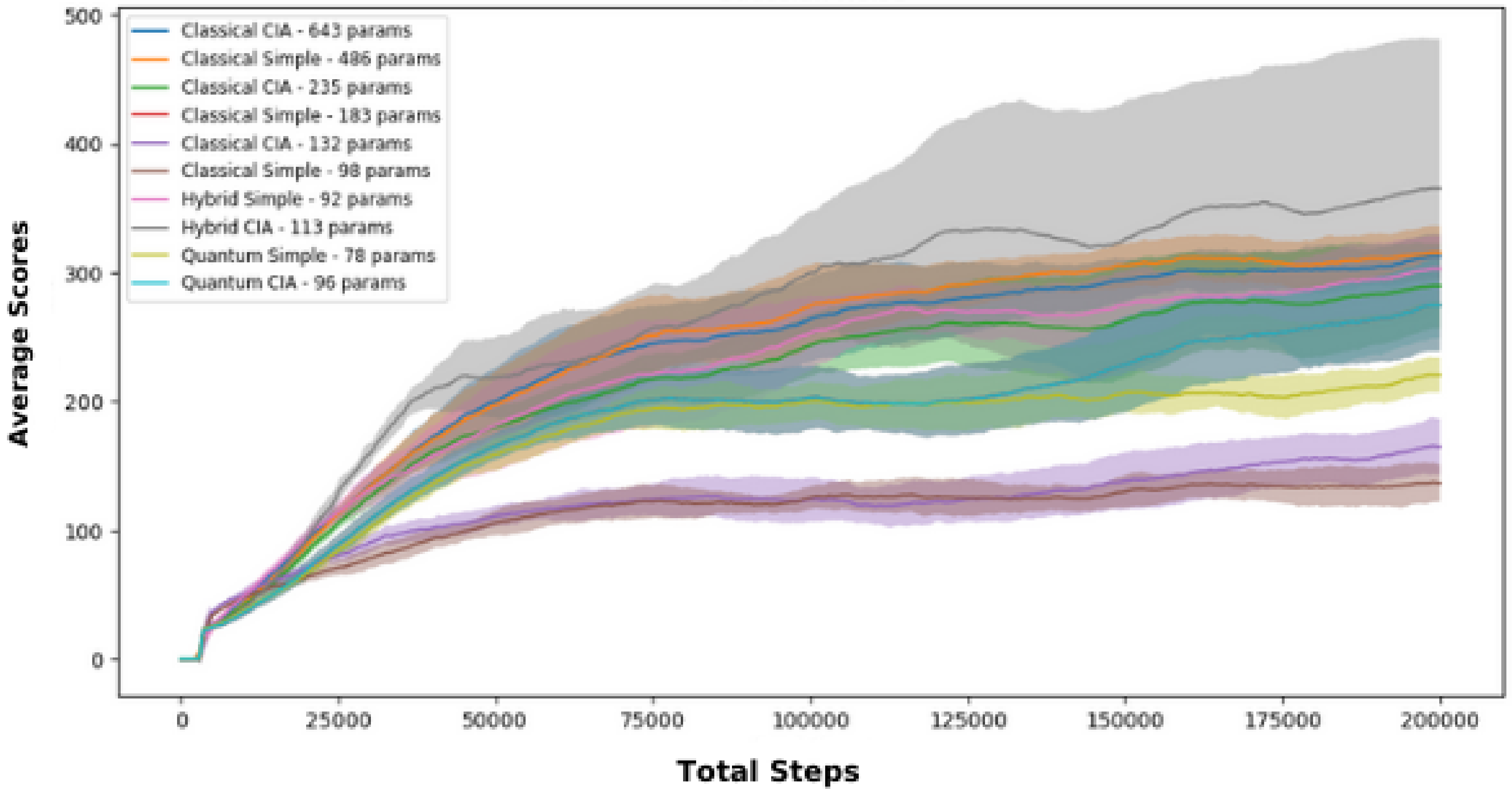


Figure 5.8: **Training steps vs. episode scores of PPO - Gaussian Noise** Peculiarly, the hybrid CIA seems to be outperforming all the other models, however, it also has a very large variance.

Table 5.6: Performance Test Scores of the PPO models tested with no noise

Network Type	Number of Parameters	Avg. Score (Out of 500)
Classical Simple	98	287.34
Classical Simple	183	458.55
Classical Simple	486	500
Classical CIA	132	278.68
Classical CIA	235	482.89
Classical CIA	643	500
Hybrid Simple	92	487.5
Hybrid CIA	113	443.44
Quantum Simple	78	495.4
Quantum CIA	96	466.16

5.2.2 Performance Scores

Each model is trained on frozen observation, Gaussian noise and blackout interference. Every model is then tested on the same interference type as they were trained on, varying the ratio of interference from 0%, 10%, 20%, 30%. The performance scored are measured in the same way as with the DQN, as the trainied models function the same way.

From Table 5.6, we can infer that using quantum layers in hybrid models and fully quantum models can help in reducing the total number of parameters used in the model without degrading performance.

In all 4 cases as referenced from Tables 5.7, 5.8 and 5.9, the simple models seem to follow a trend of lowering performance as the interference ratio increases. Such a stark trend is not noticeable in the CIAs which shows the potential improvement in resilience of this model compared to the simple models. We notice that the classical CIA and simple models with large parameters have scores of 500. We also notice that all the models have training scores above 400 except for the classical models with low number of parameters. We also note that the simple models are able to perform competitively when the interference ratio is low.

5.2.3 Robust Scores

The robust scores of the PPO models show similar trends to DQN as well and have similar scores as shown in Table 5.10. We notice that classical models with higher parameters have higher resilience. It can also be noted that the CIA helps improve the resilience of these networks. This is true across all implementations - classical, quantum and hybrid.

Table 5.7: Performance test scores of the PPO models tested with Frozen Observation noise

Network Type	**Number of Parameters**	**Avg. Score (Out of 500) for Interference Ratio**		
		10%	**20%**	**30%**
Classical Simple	98	212.4	265.56	311.54
Classical Simple	183	300.16	344.58	245.67
Classical Simple	486	450.74	389.75	315.3
Classical CIA	132	269.38	294.56	369.49
Classical CIA	235	339.59	286.75	416.83
Classical CIA	643	489.50	454.47	428.94
Hybrid Simple	92	229.89	316.83	297.78
Hybrid CIA	113	399.76	369.68	276.84
Quantum Simple	78	195.78	178.94	243.65
Quantum CIA	96	398.7	345.93	389.74

Table 5.8: Performance test scores of the PPO models tested with Gaussian noise

Network Type	**Number of Parameters**	**Avg. Score (Out of 500) for Interference Ratio of**		
		10%	**20%**	**30%**
Classical Simple	98	284.59	275.39	199.84
Classical Simple	183	309.40	337.93	356.73
Classical Simple	486	475.92	483.39	462.54
Classical CIA	132	320.50	372.14	285.69
Classical CIA	235	395.02	420.59	448.3
Classical CIA	643	494.20	478.43	369.35
Hybrid Simple	92	217.35	246.74	273.35
Hybrid CIA	113	453.63	395.64	429.35
Quantum Simple	78	262.3	194.52	187.82
Quantum CIA	96	373.24	326.25	416.21

Table 5.9: Performance test scores of the PPO models tested with Blackout noise

Network Type	**Number of Parameters**	**Avg. Score (Out of 500) for Interference Ratio of**		
		10%	**20%**	**30%**
Classical Simple	98	247.76	212.84	178.02
Classical Simple	183	297.84	385.84	274.64
Classical Simple	486	344.94	312.85	262.98
Classical CIA	132	274.59	354.24	249.24
Classical CIA	235	328.49	346.42	236.9
Classical CIA	643	445.29	430.4	386.45
Hybrid Simple	92	296.34	193.53	144.63
Hybrid CIA	113	384.39	416.6	377.21
Quantum Simple	78	182.24	165.55	115.63
Quantum CIA	96	344.56	316.74	324.22

Table 5.10: Robust scores of the PPO models

Network Type	**Number of Parameters**	**Robust Score for Noise Type of**		
		Frozen Observation	**Gaussian**	**Blackout**
Classical Simple	98	63.2%	67.4%	65.2%
Classical Simple	183	74.3%	74.2%	81.7%
Classical Simple	486	84.5%	77.3%	76.4%
Classical CIA	132	81.2%	79.1%	68.5%
Classical CIA	235	93.4%	94.1%	88.2%
Classical CIA	643	97.7%	84.2%	93.8%
Hybrid Simple	92	63.7%	72.2%	69.9%
Hybrid CIA	113	87.4%	94.3%	95.5%
Quantum Simple	78	72.1%	78.7%	68.4%
Quantum CIA	96	92.7%	87.7%	91.2%

Chapter 6

Conclusion

6.1 Discussion

The results demonstrate the feasibility of quantum-enhanced versions neural network models for DRL applications. We see that both the hybrid and the quantum models perform competitively with the classical model while utilizing far less parameters, which may even run on NISQ devices. This is true for both DQN and PPO implementations.

The results suggest that quantum networks could bring benefits over classical ones in DRL. However, this needs to be scrutinized more rigorously. More data on the training efficiencies of these models can be collected by training a more number of models. This can help quantify the sample efficiency accurately and identify which networks are the fastest to learn. However, a major problem faced while trying to scale up, is that simulating quantum networks is computationally very intensive and takes a long time.

We notice that the performance scores of the PPO tend to be higher than their corresponding performance scores from the DQN algorithm. This may be because the DQN is more intensive to train and takes longer. It was also noticed from the training scores that the DQN was not able to successfully complete training to scores of 500 and was only able to explore and stabilize around 200. This could be further improved by modifying the hyper-parameters, as DRL training is very sensitive to it.

Most importantly, we notice the two main trends from the results.

- The CIA performs better than the simple models for any implementation, with regards to sample efficiency, performance and robustness.
- Using quantum networks to process, either standalone or in hybrid networks, reduces the number of parameters required to perform as well as their classical counterparts.

6.2 Future Work

The hyper parameters used for training, such as learning rate, epsilon decay, optimize frequency and gradient clipping have a significant impact on the network's training and performance. These parameters can be optimized to provide networks which learn even quicker and play more efficiently.

The quantum networks used here all consist of 4 qubits. This could be scaled up for a bigger simulation. This may stabilize and allow the models to learn more efficiently. In this work, for the CIA models, the classical networks were directly replaced with VQCs. However, this involves measuring the outputs at every network, and losing the superposition state. Alternative methods of implementing the function of the CIA using different VQC ansatz could be explored which may maintain the superposition.

The original CIQ model from (6) makes use of previous observations along with the current state of the environment as input. This may improve the performances of the networks as they have more accurate information to process even in noisy scenarios.

The CIA can be tested under different training algorithms as well. It seems promising in the DQN and PPO implementation, and could prove to be efficient in other training algorithms as well. In this work, the agents only learned to play the Cartpole game. These models can be extended to play in other environments as well.

Similarly, improvements to the CIA can be explore. A simple extension could involve training with multiple noises at the same time. One could test whether using more Q-Value networks for every type of noise improves or depreciates performance.

Overall, the CIA model seems promising. We also note that using quantum networks in the CIA can help reduce the number of parameters required in the model. These trends show promise for new kinds of quantum model architectures to solve various problems faced in DRL.

References

[1] S. Oke, “A literature review on artificial intelligence,” *International Journal of Information and Management Sciences*, vol. 19, pp. 535–570, 12 2008.

[2] T. Hwang, “Computational power and the social impact of artificial intelligence,” *SSRN Electronic Journal*, 03 2018.

[3] V. Chauhan, S. Negi, D. Jain, P. Singh, A. K. Sagar, and A. K. Sharma, “Quantum computers: A review on how quantum computing can boom ai,” in *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pp. 559–563, 2022.

[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010.

[5] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, 01 2018.

[6] C.-H. H. Yang, I.-T. D. Hung, Y. Ouyang, and P.-Y. Chen, “Training a resilient q-network against observational interference,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 8814–8822, Jun. 2022.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” 12 2013.

[8] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, “Variational quantum circuits for deep reinforcement learning,” *IEEE Access*, vol. 8, pp. 141007–141024, 2020.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Gymnasium.” `https://github.com/openai/gym`, 2016. Accessed: 2023-11-06.

[10] N. G. Sankar, A. Khandelwal, and M. G. Chandra, "Quantum-enhanced resilient reinforcement learning using causal inference," in *2024 16th International Conference on COMmunication Systems NETworkS (COMSNETS)*, pp. 1058–1063, 2024.

[11] A. A. PIOVESANA, "An introduction to quantum reinforcement learning," 2022.

[12] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3–4, p. 219–354, 2018.

[13] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *Cambridge, MA, USA: A Bradford Book*, 2014, 2015.

[14] F. Montagna, "Quantum circuit design with reinforcement learning," 2021.

[15] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017.

[16] J. Perolat, B. De Vylder, D. Hennes, E. Tarassov, F. Strub, V. de Boer, P. Muller, J. T. Connor, N. Burch, T. Anthony, S. McAleer, R. Elie, S. H. Cen, Z. Wang, A. Gruslys, A. Malysheva, M. Khan, S. Ozair, F. Timbers, T. Pohlen, T. Eccles, M. Rowland, M. Lanctot, J.-B. Lespiau, B. Piot, S. Omidshafiei, E. Lockhart, L. Sifre, N. Beauguerlange, R. Munos, D. Silver, S. Singh, D. Hassabis, and K. Tuyls, "Mastering the game of stratego with model-free multiagent reinforcement learning," *Science*, vol. 378, pp. 990–996, Dec. 2022.

[17] Meta Fundamental AI Research Diplomacy Team (FAIR)†, A. Bakhtin, N. Brown, E. Dinan, G. Farina, C. Flaherty, D. Fried, A. Goff, J. Gray, H. Hu, A. P. Jacob, M. Komeili, K. Konath, M. Kwon, A. Lerer, M. Lewis, A. H. Miller, S. Mitts, A. Renduchintala, S. Roller, D. Rowe, W. Shi, J. Spisak, A. Wei, D. Wu, H. Zhang, and M. Zijlstra, "Human-level play in the game of diplomacy by combining language models with strategic reasoning," *Science*, vol. 378, pp. 1067–1074, Dec. 2022.

[18] X.-Y. Liu, Z. Xiong, S. Zhong, H. Yang, and A. Walid, "Practical deep reinforcement learning approach for stock trading," 2018.

[19] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," 2020.

[20] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, A. Babu, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean, "Chip placement with deep reinforcement learning," 2020.

[21] S. Mohamadi, G. Mujtaba, N. Le, G. Doretto, and D. A. Adjeroh, "Chatgpt in the age of generative ai and large language models: A concise survey," 2023.

[22] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

[23] A. Galindo and P. Pascual, "The postulates of quantum mechanics," in *Quantum Mechanics I*, pp. 33–87, Berlin, Heidelberg: Springer Berlin Heidelberg, 1990.

[24] D. P. DiVincenzo, "Quantum gates and circuits," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, p. 261–276, Jan. 1998.

[25] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," *Physical Review Letters*, vol. 113, Sept. 2014.

[26] A. Rajak, S. Suzuki, A. Dutta, and B. K. Chakrabarti, "Quantum annealing: an overview," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 381, Dec. 2022.

[27] J. Lederer, "Activation functions in artificial neural networks: A systematic overview," 2021.

[28] S. Schmidgall, J. Achterberg, T. Miconi, L. Kirsch, R. Ziaei, S. P. Hajiseyedrazi, and J. Eshraghian, "Brain-inspired learning in artificial neural networks: a review," 2023.

[29] C. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 05 1992.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[31] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2017.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, 2019.

[33] H. Wang, Y. Ding, J. Gu, Z. Li, Y. Lin, D. Z. Pan, F. T. Chong, and S. Han, "Quantumnas: Noise-adaptive search for robust quantum circuits," in *The 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA-28)*, 2022.

[34] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.