

# COMPUTING IMPLIED VOLATILITY USING QUANTUM NEURAL NETWORK



# COMPUTING IMPLIED VOLATILITY USING QUANTUM NEURAL NETWORK

## Thesis

for the purpose of obtaining the degree of Master of Science  
in Applied Mathematics  
at Delft University of Technology  
to be defended publicly on Thursday 29, February 2024 at 12 o'clock

by

**Zibo YUAN**

Faculty of Electrical Engineering, Mathematics & Computer Science,  
Delft University of Technology, Delft, the Netherlands  
born in Nanchang, China

Thesis committee:

Dr. Shuaiqiang Liu,

Prof. dr. Kees Vuik,

Prof. dr. Matthias Möller,

Dr. Fenghui Yu,

Delft University of Technology & ING Bank, supervisor

Delft University of Technology

Delft University of Technology

Delft University of Technology



An electronic version of this thesis is available at  
<http://repository.tudelft.nl/>.

*So therefore I dedicate myself, to my art, my sleep, my dreams, my labors, my sufferances, my loneliness, my unique madness, my endless absorption and hunger because I cannot dedicate myself to any fellow being.*

Jack Kerouac



# CONTENTS

<b>Abstract</b>	<b>ix</b>
<b>Acknowledgments</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Research questions . . . . .	2
1.3 Outline . . . . .	3
<b>2 Numerical methods to compute implied volatility</b>	<b>5</b>
2.1 Option pricing model . . . . .	6
2.1.1 Black-Sholes model . . . . .	6
2.1.2 Implied volatility . . . . .	7
2.2 Iterative method . . . . .	8
2.3 ANN method . . . . .	12
2.3.1 Gradient-squashing . . . . .	12
<b>3 Quantum neural network</b>	<b>15</b>
3.1 Quantum computing . . . . .	16
3.1.1 Quantum features . . . . .	16
3.1.2 Qubit . . . . .	17
3.1.3 Quantum computers . . . . .	19
3.1.4 Quantum operations and Quantum operators . . . . .	19
3.1.5 Trace and Partial trace . . . . .	20
3.1.6 Quantum circuit . . . . .	21
3.1.7 Quantum gradient . . . . .	24
3.1.8 Fidelity . . . . .	25
3.2 Quantum Neural Network . . . . .	25
3.2.1 Dissipative quantum neural network . . . . .	25
3.2.2 Parameterized variational quantum neural network . . . . .	29
<b>4 Quantum neural network to compute implied volatility</b>	<b>35</b>
4.1 Algorithm . . . . .	36
4.2 Model setup . . . . .	36
4.2.1 Data encoding . . . . .	37
4.2.2 DQNN . . . . .	38
4.2.3 PVQNN . . . . .	39

---

<b>5</b>	<b>Results and Discussion</b>	<b>45</b>
5.1	Datasets . . . . .	47
5.2	DQNN . . . . .	48
5.3	PVQNN . . . . .	50
5.3.1	Data size . . . . .	50
5.3.2	Data encoding . . . . .	53
5.3.3	Data re-uploading and Number of hidden layers . . . . .	58
5.3.4	Ansatz structure . . . . .	59
5.3.5	Optimal model . . . . .	63
5.4	Summarization . . . . .	64
<b>6</b>	<b>Conclusions and Future Research</b>	<b>67</b>
6.1	Conclusions . . . . .	67
6.2	Future research. . . . .	68
	<b>Bibliography</b>	<b>69</b>
<b>A</b>	<b>AppendixA</b>	<b>73</b>
A.1	Data size . . . . .	73
A.2	Data encoding . . . . .	74
A.3	Ansatz structure . . . . .	74
A.4	Lager dataset. . . . .	75



# ABSTRACT

Implied volatility is critical in financial markets, especially for option pricing. Traditional methods for its calculation sometimes are not well suited to some scenarios. Recent developments in neural networks have provided more efficient alternatives.

Leveraging advances in quantum computing, our research introduces quantum neural networks for computing implied volatility, assessing the feasibility and characteristics of this novel approach. We focus on two quantum neural network architectures: Dissipative Quantum Neural Networks (DQNN) and Parameterized Variational Quantum Circuits (PVQNN). DQNN, similar to classical neural networks in structure and training ease, faces challenges with quantum state outputs and data decoding, impacting performance negatively. Besides, limited by the reliance on network output states at each layer, DQNN faces challenges in implementation with the current state of quantum hardware.

In contrast, PVQNN offers a more promising solution. Compared to DQNN, PVQNN requires fewer qubits, can apply traditional optimizers to train the model, and can run on NISQ devices. This research thoroughly examines various aspects influencing PVQNN's performance, including training data characteristics, data re-uploading technology, network size, data encoding methods, and quantum circuit design. The selected PVQNN model can achieve high accuracy in implied volatility computation with  $R^2$  of approximately 0.999. In addition, we find that the PVQNN can obtain satisfactory results even with limited training data, setting it apart from traditional neural networks.

This thesis not only adopts a new model to compute implied volatility but also deepens the understanding of quantum neural networks in financial modeling. However, due to resource constraints, our experiments are conducted in simulations on traditional computers, and thus our study focuses mainly on the expressive power of QNNs rather than their operational efficiency.



## ACKNOWLEDGMENTS

*I would like to extend my deepest gratitude to all those who have made the completion of this thesis possible.*

*First and foremost, I am profoundly thankful to my thesis advisor, Dr. Shuaiqiang Liu, for his invaluable guidance, patience, and expertise. His insightful feedback and unwavering support have been pivotal to my research and academic growth.*

*I am also grateful to the members of my thesis committee, Prof. Dr. Kees Vuik, Prof. Dr. Matthias Möller, and Dr. Fenghui Yu, for their constructive criticism and suggestions that have significantly enriched my work.*

*My heartfelt appreciation goes out to my beloved friends, whose encouragement, intellectual discussions, and companionship provided a much-needed respite from the demands of research. Your support has been a pillar of strength and a source of inspiration.*

*To my family, especially my parents, your endless love, understanding, and support have been my foundation. This achievement is as much yours as it is mine, for you have been my unwavering rock throughout this journey.*

*I also wish to take a moment to acknowledge my own efforts and resilience. The dedication, hard work, and perseverance I invested in this project have been a journey of personal growth and self-discovery. Navigating through challenges and celebrating each small victory has been an enriching experience that I will cherish forever.*

*This journey, filled with challenges and rewards, could not have been navigated without the support and encouragement of each one of you, especially who I hold dear. Thank you from the bottom of my heart for being part of this significant phase of my life.*

*Last but not least, when you need time to heal, try Lana Del Rey's songs and the nature of the Netherlands. They've been the Felix Felicis for me in this period.*

Zibo

Delft, February 2024



# 1

## INTRODUCTION

### 1.1 MOTIVATION

Implied volatility is a crucial component in the financial markets, serving as a key attribute in option pricing. It signifies the anticipated volatility of an option's price and serves as an indicator of market trends. Typically included as an essential parameter in option pricing, the exact solution for implied volatility often remains elusive, necessitating numerical methods for approximation.

Several iterative methods are commonly used for their distinct advantages in computational. The Newton-Raphson's method is favored for its rapid convergence when the derivative of a function is known, making it efficient for estimating implied volatility. Brent's Method, merging the bisection method and inverse quadratic interpolation, excels in more complex situations where Newton-Raphson's method may fail. The bisection method, known for its simplicity, is reliable but slower, and the Secant method is an alternative when derivatives are hard to compute. Each of these methods, while effective, has its limitations, urging ongoing research for more efficient and accurate solutions.

Another numerical method introduced to computing implied volatility is the Artificial Neural Networks (ANNs). Since the rise of neural networks has revolutionized many fields, including finance. Their ability to extract features and identify patterns has led to significant advancements. For instance, deep neural networks have been effectively utilized in calibrating pricing models [1–3] and hedging derivative portfolios[4–6]. Notably, ANNs have demonstrated remarkable proficiency in computing implied volatility[7]. Compared to the previously mentioned iterative approach, not only is there a relatively high model accuracy but also a much faster computation of the output implied volatility, stressing its potential in complex financial computations.

Apart from these developments, quantum computing has risen, marking a new era in computational capabilities[8]. Its application in finance, particularly in derivative pricing through amplitude estimation, has shown a quadratic speed-up compared to traditional Monte Carlo methods[9, 10]. This up-and-coming field offers a new perspective on financial problem-solving, its potential has been highlighted in a comprehensive overview of its applications in mathematical finance[11].

Besides, a frontier of research that combines the principles of quantum computing with deep learning is particularly intriguing. For example, networks like quantum neural networks[12–14] and quantum convolutional neural networks[15, 16] represent a fusion of deep neural networks’ robust generalization capabilities with the unique computational advantages of quantum computing. Moreover, research into quantum neural networks has sparked considerable interest due to their potential applications in real-world issues. Such as solving nonlinear differential equations, a foundational element in many scientific and engineering problems[17]. Additionally, their application in predicting drug responses showcases the potential of quantum neural networks in the field of biomedicine and pharmaceuticals[18].

It is worth mentioning here that some researchers are also exploring the use of quantum neural networks (QNNs) in areas closely aligned with our research focus, as demonstrated in the work by Sakuma [19] and Eric et al. [20]. In the work of Sakuma, he used a type of quantum neural network to approximate the relationship between the strike price and implied volatility, demonstrating that using a QNN to compute implied volatility is a viable pathway. Other than that, in the work of Eric et al., they successfully utilized the unique characteristics of QNNs in data encoding to convert financial time series data into a sequence of density matrices and used them as inputs to a quantum neural network for stock price prediction.

However, despite these advancements, quantum neural networks are still in a newborn stage of development. Many aspects of their operational paradigms and characteristics remain under-explored. While their potential to surpass traditional computing methods in certain aspects is promising, there are still many difficulties in empirical validation and confirmation of this superiority.

Given the proven efficacy of artificial neural networks in addressing the challenge of implied volatility, the related successful applications of quantum computing in finance, and the rapid advances in the development of quantum neural networks, our research is motivated by these very techniques. We aim to explore the advantages of quantum neural networks in tackling the complexities of implied volatility.

Our research is driven by the demonstrated effectiveness of ANNs in addressing the challenge of implied volatility. This, coupled with the explorations of quantum computing within the finance sector, and the speedy progress in the development of quantum neural networks, forms the foundation of our investigation. We are particularly interested in exploring how quantum neural networks might enhance the calculation of implied volatility. Our goal is to unravel the potential benefits and efficiencies that quantum neural networks could bring to the domain of financial modeling, especially in the computation of implied volatility.

## 1.2 RESEARCH QUESTIONS

In our research, we have investigated two distinct architectures of quantum neural networks, Dissipative Quantum Neural Networks (DQNNs) and Parameterized Variational Quantum Neural Networks (PVQNNs), conducting a thorough exploration to identify key factors that influence the performance of these quantum models in computing implied volatility. This investigation is guided by two primary objectives:

1. To assess the feasibility of quantum neural networks for managing the complexities in implied volatility calculations. This includes an exploration of their unique properties and how these distinguish them from conventional computational methods.
2. To identify and analyze the factors that impact the effectiveness of quantum neural networks in the specific context of implied volatility. This involves a deeper examination of network architecture, training methodologies, and the adaptation of these networks to solve related financial problems.

It is worth noting that due to the practical implementation challenges associated with DQNN, our investigation primarily focuses on PVQNN for the second research question. These models, while being at the forefront of quantum computing research, present a unique set of challenges and opportunities in the field of financial modeling. We aim to explore how the intrinsic characteristics of PVQNN models, such as their parameterization and quantum circuit design, impact their applicability and performance in modeling and predicting implied volatility. By doing so, we hope to contribute valuable insights into the practical applications of quantum neural networks in solving complex financial problems, paving the way for more advanced and efficient computational approaches in finance.

## 1.3 OUTLINE

Our paper is organized as follows: Chapter 2 and Chapter 3 provide a detailed background for our work. Chapter 2 begins with an overview of the Black-Scholes asset pricing model, setting the stage for our investigation into implied volatility computation. We then describe how this problem can be solved using iterative and ANN methods. This is followed by an introduction to the fundamentals of quantum computing in Chapter 3, providing the necessary quantum background for understanding our approach. Subsequently, we delve into a comprehensive discussion of the two quantum neural network architectures under study, detailing their structures and training processes.

In Chapter 4, we outline the algorithmic framework of our research, including the settings and variables relevant to our model. This chapter focuses on examining how different factors influence the model's output, leading to the selection of the optimal model configuration for our problem. The results of these numerical experiments, along with in-depth analyses, are presented in Chapter 5.

The final section of our paper offers a summary of our research findings and insights. It also casts a forward-looking perspective, discussing potential future research directions in the field, thereby framing our work within the broader context of quantum computing and financial modeling.





# 2

## NUMERICAL METHODS TO COMPUTE IMPLIED VOLATILITY

*In this chapter, we provide detailed background information for our research question, computing implied volatility. First, the introduction of implied volatility is given in Section 2.1. Then, in Section 2.2, some iterative numerical methods for computing implied volatility are introduced. Lastly, in Section 2.3, we briefly describe how to use ANN to solve our research question.*

## 2.1 OPTION PRICING MODEL

Options, as significant derivative assets in financial markets, play a crucial role in enhancing market dynamics. They not only facilitate increased participation by offering diverse investment strategies but also contribute to the expansion of capital availability. By hedging and speculation, options encourage a broader spectrum of investors to engage in financial markets. Moreover, options help improve the liquidity of other less active contracts, making the overall market more efficient and responsive.

Given their importance, the accurate pricing of options has emerged as a pivotal area of research in financial economics. Accurate option pricing models are essential for both investors and issuers to assess risk, make informed decisions, and ensure fair value trading. In this section, we will focus on the Black-Scholes model, one of the most famous and widely used models for pricing European options. The model particularly highlights the significance of implied volatility in option pricing. Furthermore, we will also briefly discuss some traditional numerical methods which are important for calculating implied volatility.

### 2.1.1 BLACK-SHOLES MODEL

The Black-Scholes model, established by Fischer Black and Myron Scholes in 1973 [21], stands as a cornerstone in the realm of financial economics, particularly for pricing European options. This model introduced a systematic approach to option valuation, containing various market factors into a coherent theoretical framework.

In financial markets, the return of an asset over a short interval can be expressed as,

$$\text{Return} = \frac{\text{asset price at future} - \text{asset price at present}}{\text{Stock at present}} = \frac{S(t + \Delta t) - S(t)}{S(t)}.$$

Here,  $S(t)$  represents the price of the asset at time  $t$ , and  $S(t + \Delta)$  is the price at a future time. However, it's crucial to note that in reality, the daily return  $dS(t)$  often behaves unpredictably, like noise, which corresponds to a stochastic process. This observation is crucial for understanding the foundation of the Black-Scholes model.

The most commonly used stochastic process for modeling the price of a non-dividend-paying asset,  $S(t)$ , is the Geometric Brownian Motion (GBM),

$$dS_t = \mu S_t dt + \sigma S_t dW_t^P. \quad (2.1)$$

In this equation,  $W_t^P$  represents a Wiener process (also known as Brownian motion) under the real-world probability measure  $P$ , which captures the random movement in asset prices. The term  $\mu$  is the drift coefficient, while  $\sigma$  represents the implied volatility of the asset, a measure of how much the asset's price is expected to fluctuate.

Therefore, in a friction-free market, the price of a European option's underlying stock can be modeled using the stochastic differential equation described by GBM as mentioned in Equation (2.1). If this process begins with the construction of a risk-neutral portfolio, characterized by the Wiener process under the risk-neutral measure  $Q$ . Accordingly, Equation (2.1) can be reformulated as:

$$dS_t = rS_t dt + \sigma S_t dW_t^Q. \quad (2.2)$$

Subsequently, utilizing Itô's Lemma allows for the derivation of an arbitrage-free option pricing model. This is represented by a partial differential equation (PDE), known as the Black-Scholes equation,

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0. \quad (2.3)$$

Here,  $t$  denotes time,  $S$  is the stock price,  $V$  is the option price,  $r$  is the risk-free interest rate, and  $\sigma$  represents the volatility of the stock. Given that we are dealing with European options, which can only be exercised at maturity, there is an associated boundary condition. For instance, a vanilla European call option with a strike price  $K$  and expiry date  $T$  will have the following condition at maturity,

$$V(t = T, S) = (S - K)^+ = \max\{0, S - K\} = \begin{cases} S - K, & S > K \\ 0, & S \leq K \end{cases}$$

Accordingly, for a vanilla European call option, the Black-Scholes equation becomes,

$$\begin{cases} \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0 & 0 \leq S \leq \infty, 0 \leq t \leq T \\ V|_{t=T} = (S - K)^+ \end{cases} \quad (2.4)$$

This equation can be transformed into the form of a heat equation, and yielding an analytical solution,

$$V(t, S) = SN(d_1) - Ke^{-r\tau} N(d_2), \quad (2.5)$$

$$d_1 = \frac{\log(S/K) + (r - 0.5\sigma^2)\tau}{\sigma\sqrt{\tau}}, d_2 = d_1 - \sigma\sqrt{\tau},$$

where  $\tau := T - t$  is the time to expiration, and  $N(\cdot)$  denotes the cumulative distribution function of the standard normal distribution. This solution is commonly represented as  $V = BS(\sigma, r, \tau, K, S)$ , indicating the formula of the Black-Scholes model for a European call option's value.

### 2.1.2 IMPLIED VOLATILITY

Black-Scholes formula contains the strike price  $K$  and expiry date  $T$ , which are specified in the option contract, while the interest rate  $r$  and stock price  $S$  are observable in the market. The remaining variable to determine is the implied volatility  $\sigma$ . Therefore, implied volatility is a central factor in options trading and is crucial for pricing, trading strategies, and risk assessment. This is a key parameter that reflects the market's estimate of how volatile an asset will be in the future

A similar concept to implied volatility is historical (or realized) volatility, which measures the degree of variation in the price of a financial instrument over time. It's typically calculated by taking the standard deviation of daily logarithmic returns over a specific historical period.

Historical volatility reflects the volatility of a stock in the past, while implied volatility captures future volatility. Therefore, implied volatility is often more relevant for options traders, as it helps in assessing the likelihood of a stock reaching a specific price by the option's expiry.

## 2.2 ITERATIVE METHOD

In the Black-Scholes model, the implied volatility of an option is assumed to be constant. This assumption allows for estimating the implied volatility based on historical market data. The market option price  $V^{mkt}$  can be used to calculate an option's implied volatility by solving the inverse of the Black-Scholes formula,

$$\sigma = BS^{-1}(V^{mkt}, \tau, S, r, k). \quad (2.6)$$

There is no analytical solution for this equation; however, it can be solved using various iterative root-finding algorithms to solve this,

$$f(\sigma) = BS(\sigma) - V^{mkt} = 0. \quad (2.7)$$

Here are some of the most frequently used numerical methods:

### 1. Newton-Raphson's method

This method is favored for its rapid convergence. It utilizes the derivative of the Black-Scholes formula with respect to volatility and iteratively adjusts the volatility estimate to match the market price. A good initial estimate is crucial, as a poor estimate can slow the convergence.

---

#### Algorithm 1 Newton-Raphson Method for Implied Volatility

---

**Input:** Initial estimate  $\sigma_0$ , tolerance  $\epsilon$

**Output:** Implied Volatility  $\sigma$

**while**  $|f(\sigma_n)| > \epsilon$  **do**

$$\sigma_{n+1} \leftarrow \sigma_n - \frac{f(\sigma_n)}{f'(\sigma_n)}$$

**end while**

**return**  $\sigma_{n+1}$

---

$f'(\sigma)$  is the derivative of  $f(\sigma)$  with respect to  $\sigma$ .

### 2. Bisection Method

It starts with two initial estimates that bracket the true implied volatility and then iteratively narrows down the range. It is guaranteed to converge but may take more iterations.

### 3. Secant Method

This is a derivative-free method and can be seen as a variant of the Newton-Raphson method. It replaces the derivative in the Newton-Raphson formula with an approximation based on the function values at two points. It's more robust than Newton-Raphson but is less robust than the bisection method.

### 4. Inverse interpolation Method

This method involves interpolating the inverse of the function to avoid complex values, resulting in inverse quadratic interpolation. While it converges faster asymptotically than the secant method, it can be less reliable when the initial estimates are not close to the root.

**Algorithm 2** Bisection Method for Implied Volatility

---

**Input:** Initial brackets  $\sigma_{low}$ ,  $\sigma_{high}$ , tolerance  $\epsilon$   
**Output:** Implied Volatility  $\sigma$   
**while**  $|\sigma_{high} - \sigma_{low}| > \epsilon$  **do**  
     $\sigma_{mid} \leftarrow \frac{\sigma_{low} + \sigma_{high}}{2}$   
    **if**  $f(\sigma_{mid}) \cdot f(\sigma_{low}) < 0$  **then**  
         $\sigma_{high} \leftarrow \sigma_{mid}$   
    **else**  
         $\sigma_{low} \leftarrow \sigma_{mid}$   
    **end if**  
**end while**  
**return**  $\sigma_{mid}$

---

**Algorithm 3** Secant Method for Implied Volatility

---

1: **Input:** Initial estimate  $\sigma_0$ ,  $\sigma_1$ , tolerance  $\epsilon$   
2: **Output:** Implied Volatility  $\sigma$   
3: **while**  $|f(\sigma_n)| > \epsilon$  **do**  
4:      $\sigma_{n+1} \leftarrow \sigma_n - f(\sigma_n) \frac{\sigma_n - \sigma_{n-1}}{f(\sigma_n) - f(\sigma_{n-1})}$   
5: **end while**  
6: **return**  $\sigma_{n+1}$

---

## 5. Brent's Method

A combination of bisection, secant, and inverse quadratic interpolation, Brent's method is often preferred for its balance between robustness and speed. It adjusts its strategy based on the situation and usually converges faster than the bisection method.

These methods often present a trade-off between speed, robustness, and computational complexity. For example, while the Bisection Method is robust enough, it tends to have a longer convergence time. The choice of method typically depends on the specific requirements and constraints of the given problem.

---

**Algorithm 4** Inverse Interpolation Method for Implied Volatility
 

---

```

1: Input: Initial estimate  $\sigma_0, \sigma_1, \sigma_2$ , tolerance  $\epsilon$ 
2: Output: Implied Volatility  $\sigma$ 
3: while  $|f(\sigma_0)| > \epsilon$  do
4:    $f_0, f_1, f_2 \leftarrow f(\sigma_0), f(\sigma_1), f(\sigma_2)$ 
5:    $\sigma_{new} \leftarrow \text{INVERSE QUADRATIC INTERPOLATION}(\sigma_0, \sigma_1, \sigma_2, f_0, f_1, f_2)$ 
6:   Sort  $\sigma_0, \sigma_1, \sigma_2$  based on  $|f_0|, |f_1|, |f_2|$ 
7:    $\sigma_0 \leftarrow \sigma_{new}$ 
8:    $\sigma_1 \leftarrow \sigma_0$ 
9:    $\sigma_2 \leftarrow \sigma_1$ 
10: end while
11: return  $\sigma_0$ 
12: procedure INVERSE QUADRATIC INTERPOLATION( $\sigma_0, \sigma_1, \sigma_2, f_0, f_1, f_2$ )
13:    $q_0 \leftarrow \frac{f_0}{f_1}; q_1 \leftarrow \frac{f_2}{f_1}; r_0 \leftarrow \frac{f_0}{f_2}; r_1 \leftarrow \frac{f_1}{f_2}$ 
14:    $s_0 \leftarrow \sigma_0 \cdot q_0 \cdot (q_0 - q_1) \cdot (q_0 - r_0)$ 
15:    $s_1 \leftarrow \sigma_1 \cdot r_0 \cdot (r_0 - q_1) \cdot (r_0 - q_0)$ 
16:    $s_2 \leftarrow \sigma_2 \cdot q_1 \cdot (q_1 - q_0) \cdot (q_1 - r_0)$ 
17:    $P \leftarrow s_0 + s_1 + s_2$ 
18:    $Q \leftarrow (q_0 - q_1) \cdot (q_0 - r_0) \cdot (r_0 - q_1)$ 
19:    $\sigma_{new} \leftarrow P/Q$ 
20:   return  $\sigma_{new}$ 
21: end procedure

```

---

**Algorithm 5** Brent's Method for Implied Volatility

---

```

1: Input: Initial Bracketing interval  $[\sigma_a, \sigma_b]$ , tolerance  $\epsilon$ 
2: Output: Implied Volatility  $\sigma$ 
3:  $\sigma_c \leftarrow \sigma_a$ 
4:  $f_a, f_b \leftarrow f(\sigma_a), f(\sigma_b)$ 
5:  $f_c \leftarrow f_a$ 
6:  $\text{flag } mflag \leftarrow \text{True}$ 
7: while  $|f_b| > \epsilon$  and  $|b - a| > \epsilon$  do ▷ Check for convergence
8:   if  $f_a \neq f_c$  and  $f_b \neq f_c$  then
9:      $s \leftarrow \text{INVERSE QUADRATIC INTERPOLATION}(\sigma_a, \sigma_b, \sigma_c, f_a, f_b, f_c)$ 
10:   else
11:      $s \leftarrow \sigma_b - f_b \cdot \frac{\sigma_b - \sigma_a}{f_b - f_a}$  ▷ Use secant method
12:   end if
13:   Condition 1:  $s$  not between  $(3\sigma_a + \sigma_b)/4$  and  $\sigma_b$ 
14:   Condition 2: ( $mflag$  is True and  $|s - \sigma_b| \geq |\sigma_b - \sigma_c|/2$ )
15:   Condition 3: ( $mflag$  is False and  $|s - \sigma_b| \geq |\sigma_c - \sigma_d|/2$ )
16:   Condition 4: ( $mflag$  is True and  $|\sigma_b - \sigma_c| < \epsilon$ )
17:   Condition 5: ( $mflag$  is False and  $|\sigma_c - \sigma_d| < \epsilon$ )
18:   if Condition 1 or Condition 2 or Condition 3 or Condition 4 or Condition 5 then
19:      $s \leftarrow (\sigma_a + \sigma_b)/2$  ▷ Use bisection method
20:      $mflag \leftarrow \text{True}$ 
21:   else
22:      $mflag \leftarrow \text{False}$ 
23:   end if
24:    $\sigma_d \leftarrow \sigma_c, \sigma_c \leftarrow \sigma_b$ 
25:   if  $f_a \cdot f_s < 0$  then
26:      $\sigma_b \leftarrow s$ 
27:   else
28:      $\sigma_a \leftarrow s$ 
29:   end if
30:   if  $|f_a| < |f_b|$  then
31:     Swap  $\sigma_a$  and  $\sigma_b$ 
32:   end if
33: end while
34: return  $\sigma_b$ 

```

---

## 2.3 ANN METHOD

The recent advancements in machine learning, particularly in artificial neural networks (ANNs), have opened new avenues for addressing complex financial problems, such as the calculation of implied volatility. The study by Liu et al. [7] is a testament to the potential of neural networks in this specialized domain. Their work demonstrates the efficacy of neural networks in providing solutions to computing the implied volatility.

Using ANN to compute implied volatility first starts with data collection and preprocessing, input data such as the stock price  $S$ , strike price  $K$ , option price  $V$ , interstate  $r$ , and expiry time  $T$  from the Black-Scholes formula are fed into the network. The neural network's output is the implied volatility  $\sigma$ .

Then we need to design the ANN with multiple layers, including input, hidden, and output layers. Each layer is made up of numerous neurons. A neuron, which involves learnable weights and biases, is the fundamental unit of ANNs, providing the linearity to the model. The number of hidden layers and neurons in each layer can vary based on the complexity of the task. Besides, choosing appropriate activation functions (like ReLU, sigmoid, or tanh) for the neurons is also very important, particularly in the hidden layers, to introduce non-linearity.

After that, the ANN is trained to learn the relationship, which is the Equation (2.6), between inputs  $(S, K, V, T, r)$  and the target output  $\sigma$  by optimization algorithms (like stochastic gradient descent). It can update the weights in the network through back-propagation, minimizing the difference between the ANN's predictions and actual target implied volatilities.

In Liu's work [7], the ANN method not only ensures the accuracy of the model but also provides a significant improvement in the time used to output the implied volatility compared to the iterative approach. It highlights the potential for more accurate and efficient computation methods in the field. As machine learning technology continues to advance, its integration into financial models like the Black-Scholes framework is likely to yield increasingly sophisticated and reliable tools for financial analysis and decision-making.

### 2.3.1 GRADIENT-SQUASHING

In practical scenarios, the sensitivity of an option's price to changes in volatility, known as Vega,  $\frac{\partial V}{\partial \sigma}$ , can be exceedingly small. This characteristic poses a significant challenge during the optimization process. Specifically, when computing the gradient of ANN model output  $\sigma$  with respect to the model input  $V$ , it is inversely related to the Vega. This inverse relationship can lead to a phenomenon known as gradient explosion, where the gradient values become excessively large. Such gradient explosions can induce significant prediction errors and cause convergence issues in the model, undermining its predictive accuracy and stability.

To mitigate this issue, we can adopt the gradient-squash approach proposed by Liu et al. [7]. This technique is particularly used for in-the-money (ITM) call options, where the market price  $V$  exceeds the strike price  $S$ . An ITM call option's price comprises two components: the intrinsic value and the time value. The intrinsic value, calculated as  $\max(S - Ke^{-rt}, 0)$ , represents the profit that could be realized if the option were exercised immediately. We modify the option's price by subtracting this intrinsic value, as follows,



$$\tilde{V} = V - \max(S - Ke^{-r\tau}, 0).$$

This adjustment effectively isolates the time value of the option, which is more sensitive to changes in volatility and other market factors. Furthermore, to address the steepness of the gradient that still might be present after this initial modification, we apply a logarithmic transformation. This transformation effectively 'flattens' the gradient, reducing the risk of gradient explosion by smoothing out sharp changes in the gradient values.



# 3

## 3

## QUANTUM NEURAL NETWORK

*In this chapter, we provide some background information for quantum neural networks. First, in Section 3.1, some basic knowledge of quantum computing is introduced. Then, in Section 3.2, we detail the structure and training process of DQNN and PVQNN respectively.*

### 3.1 QUANTUM COMPUTING

Ever since the establishment of the modern theory of quantum mechanics in the early 1920s, generations of scientists have been diligently engaged in the mysterious realm of quantum phenomena, striving to refine and advance the foundational theories. This relentless pursuit has led to the birth of quantum computing, a remarkable field with the primary objective of harnessing mathematical tools to explore and comprehend quantum mechanics at a level of precision that transcends human intuition. The aspiration is to bridge the gap between the profoundly counterintuitive nature of quantum mechanics and our logical comprehension of the universe [22].

For example, the state of a quantum mechanical system can be described by a vector  $|\Psi\rangle$ , known as the state vector, belonging to a Hilbert space  $\mathcal{H}$ , and the change in a quantum state over time follows a linear differential equation, Schrödinger equation [23]. This equation, as formalized by Erwin Schrödinger, takes the form,

$$i\hbar \frac{d}{dt} |\Psi(t)\rangle = \hat{H} |\Psi(t)\rangle, \quad (3.1)$$

where  $\hbar$  is a constant and  $\hat{H}$  represents the Hamiltonian operator. It's a Hamiltonian matrix mathematically, describing the total energy of the quantum system, which contributes significantly to understanding and predicting the behavior of quantum systems in diverse physical scenarios.

In addition, quantum computing has a wider range of applications in quantum mechanics, which we will describe in detail in the following part.

#### 3.1.1 QUANTUM FEATURES

Unlike classical computation, quantum computing can be seen as a paradigm that follows the laws of quantum mechanics to govern quantum information units for computation, which brings some unique features to quantum computing.

##### QUANTUM SUPERPOSITION

One of the core features of quantum computing is the superposition, a fundamental principle of quantum mechanics. In a quantum system, the mode of motion for a quantum system is called a state. Unlike classical systems, where the states of systems are precisely determined, in the microscopic realm, quantum states of motion remain indeterminate and inherently probabilistic. This fundamental property enables particles to exist in a superposition of multiple states simultaneously, vastly expanding the computational potential of quantum computers.

##### QUANTUM ENTANGLEMENT

Another remarkable attribute of quantum computing is quantum entanglement. When two particles become entangled, their properties become interconnected in such a way that the behavior of one particle influences the state of the other, regardless of the physical distance that separates them. This phenomenon defies classical intuitions, as changes in one particle's state instantaneously affect the other, no matter how far apart they are. Quantum entanglement not only gives rise to correlations between particles that are impossible to achieve in classical systems but also enhances the complexity of quantum

states. This property empowers quantum algorithms to exploit the structural characteristics of problems and inputs, possibly outperforming classical counterparts.

### QUANTUM PARALLELISM

Quantum parallelism is a striking advantage offered by quantum computing, rooted in the superposition and entanglement properties of qubits. In practical terms, this means that quantum computers can represent numbers in an exponential form and execute unitary operations on all these numbers simultaneously. This innate parallelism grants quantum computers the potential to tackle complex problems at a pace exponentially faster than classical counterparts, offering breakthroughs in fields such as cryptography, optimization, and simulations.

Essentially, quantum computing leverages properties such as superposition, entanglement, and parallelism to usher in another era of computation, opening the door to solving problems that were once deemed intractable and revolutionizing fields such as cryptography, optimization, and simulation.

#### 3.1.2 QUBIT

In classical computation, the 'bit' stands as both a fundamental unit of information and the building block of binary numbers, functioning as the ultimate measure of information granularity. Drawing from Shannon's information theory, a 'bit' can be used to describe a signal's potential states as 0 and 1.

Similarly, in quantum computing, we have the 'qubit' as the foundational component. For a qubit, it can be in a pure state, representing a superposition of possibilities, or in a mixed state, which characterizes a statistical ensemble of pure states. This mixed state often arises when qubits are entangled with one another, allowing for the description of their joint quantum state.

### BRA-KET NOTATION

We often use Bra-ket notation, also known as Dirac notation, for describing quantum states. Here the  $|\psi\rangle$  is the ket notation for  $\psi$  state, and the complex conjugate of  $\psi$  state  $\langle\psi|$  is the corresponding bra notation.

As we use 0 or 1 to represent bits' states, we use  $|0\rangle$  and  $|1\rangle$  to denote the computational basis states for qubits. When describing the pure state of a qubit  $|\psi\rangle$ , we often refer to it as being in a superposition, represented as,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad (3.2)$$

where the coefficient numbers  $\alpha$  and  $\beta$ , which we call amplitude, are normally complex numbers.

In classical computation, we can easily check a bit to determine whether it is in the state '0' or '1'. On the contrary, It is significant to note that we cannot examine a qubit to determine its amplitude. Instead, when we measure a qubit, we obtain a  $|0\rangle$  state with probability  $\alpha^2$  or a  $|1\rangle$  state with probability  $\beta^2$ . Therefore, these probabilities must sum to 1:  $\alpha^2 + \beta^2 = 1$ .

erathHowever, it is also possible for a quantum system to be in a statistical ensemble of different pure states, a mixed state  $\rho$ . We use an operator, also called the density matrix  $\rho$ , to express it,

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|, \quad (3.3)$$

where  $p_i$  is the probability associated with each pure state  $|\psi_i\rangle$ , indicating the likelihood that the system will be observed in a different state.

### 3

#### VECTOR FORM

Mathematically, we define the quantum computational basis states as a vector in a two-dimensional Hilbert space,

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix}; |1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Therefore, according to the Equation (3.2), we can define the pure state of a qubit as a vector,

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \alpha^2 + \beta^2 = 1.$$

More generally, for an n-qubit system, we use the notation  $|x_1x_2\dots x_n\rangle$ , where each  $x_i$  can be either '0' or '1', as the computational basis state. The state of this system can also be represented as a vector in a  $2^n$  dimensional Hilbert space.

#### BLOCH SPHERE

Geometrically, a qubit's state can be effectively visualized using the Bloch sphere, as depicted in Figure 3.1. The Bloch sphere is a spherical representation of a qubit's state, providing an intuitive way to grasp its properties.

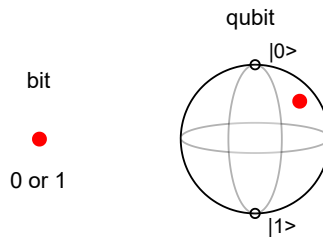


Figure 3.1: Bloch Sphere.

The pure state of a qubit,  $|\psi\rangle$ , can also be expressed in terms of two angles,  $\theta$  and  $\phi$ , as follows,

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right)|1\rangle. \quad (3.4)$$

Here,  $\theta$  lies in the range of  $0 \leq \theta \leq \pi$ , and  $\phi$  ranges from 0 to  $2\pi$ . These angles correspond to points on the surface of the Bloch sphere.

The points on the surface of the Bloch sphere represent the pure states of a single qubit system. In contrast, the interior of the sphere corresponds to mixed states. The Bloch sphere provides a geometric visualization that greatly simplifies the understanding of qubit states and their dynamics.

### 3.1.3 QUANTUM COMPUTERS

Quantum computers are designed with different physical implementations and use qubits to perform quantum computing, each with unique characteristics and applications.

1. Superconducting quantum computer

Superconducting quantum computers are a leading type of quantum device that harness superconducting qubits, artificial atoms capable of existing in multiple quantum states simultaneously. These qubits are manipulated using microwave pulses, and they are celebrated for their scalability. Companies like IBM and Google are actively developing superconducting quantum computers, aiming to enhance their performance and unlock their potential for a wide range of applications.

2. Photonic quantum computers

In these Photonic quantum computers, information is encoded and processed using photons, the particles of light. Photonic quantum computers have the advantage of extremely low error rates and can be more robust against certain types of errors. They are particularly promising for applications in quantum communication and quantum cryptography.

3. Quantum annealers

Quantum annealers represent a distinct category. These machines, such as those developed by D-Wave, are optimized for solving specific optimization problems. They employ quantum annealing to find the global minimum of a cost function, which has applications in fields like finance, logistics, and artificial intelligence.

The current state of quantum computers is referred to as the noisy intermediate-scale quantum (NISQ) devices and can be seen as the first working models of quantum computers. They are called 'noisy' because they suffer from errors and decoherence, limiting their performance and reliability. NISQ devices typically have a moderate number of qubits, on the order of tens to hundreds. While they are not yet capable of running fault-tolerant quantum algorithms, they hold great promise for practical applications, including quantum optimization, quantum machine learning, and quantum simulations.

### 3.1.4 QUANTUM OPERATIONS AND QUANTUM OPERATORS

In quantum mechanics, operators are mathematical entities that play a crucial role in representing physical processes and observables associated with quantum systems. Unlike their counterparts in classical mathematics, quantum operators capture the dynamic evolution and measurable properties of quantum systems, like position, momentum, or energy. One fundamental application of quantum operators is describing changes in the state of a quantum system induced by physical processes.

Consider a quantum system that is initially in a state  $\rho$ , then entangle it with its environment  $\rho_{\text{env}}$ , and the whole quantum system can be represented as an outer product

state  $\rho \otimes \rho_{\text{env}}$ . After a series of quantum transformations, we obtain a new quantum system, and this change can be expressed in terms of the quantum operator  $U$ . To extract the reduced quantum state of the evolved system, we perform a partial trace over the environment, see the detail of the partial trace in Section 3.1.5,

$$\mathcal{E}(\rho) = \text{tr}_{\text{env}} [U(\rho \otimes \rho_{\text{env}})U^\dagger]. \quad (3.5)$$

Here,  $\mathcal{E}$  denotes the quantum operation, providing a comprehensive description of quantum system evolution in various scenarios [22]. The expression can be further elaborated using the Kraus decomposition,

3

$$\begin{aligned} \mathcal{E}(\rho) &= \sum_k \langle e_k | U[\rho \otimes |e_0\rangle\langle e_0|] U^\dagger | e_k \rangle \\ &= \sum_k A_k \rho A_k^\dagger. \end{aligned} \quad (3.6)$$

Here, let  $\rho_{\text{env}} = |e_0\rangle\langle e_0|$  be the initial state of the environment and  $A_k = \langle e_k | U | e_0 \rangle$ , with  $|e_k\rangle$  forming an orthonormal basis for the environment's state space. This representation is known as the Kraus decomposition, where  $A_k$  are Kraus operators satisfying the completeness condition,

$$\sum_k A_k A_k^\dagger = 1,$$

### 3.1.5 TRACE AND PARTIAL TRACE

#### 1. Trace for quantum computing

The trace of a matrix, denoted as  $\text{tr}(A)$ , is defined as the sum of its diagonal elements. This mathematical operation exhibits several essential properties:

- **Cyclicity:** The trace is cyclic, meaning that for any matrices  $A$  and  $B$ ,  $\text{tr}(AB) = \text{tr}(BA)$ .
- **Linearity:** The trace is linear, which implies that for matrices  $A$  and  $B$  and a complex number  $z$ ,  $\text{tr}(A+B) = \text{tr}(A) + \text{tr}(B)$  and  $\text{tr}(zA) = z \times \text{tr}(A)$ .

If  $|\psi\rangle$  is a state vector and  $A$  is an arbitrary quantum operator, we can extend  $|\psi\rangle$  to create an orthonormal basis  $|i\rangle$ , with  $|\psi\rangle$  as the first element. The trace of the product  $A|\psi\rangle\langle\psi|$  can be calculated as:

$$\begin{aligned} \text{tr}(A|\psi\rangle\langle\psi|) &= \sum_i \langle i | A | \psi \rangle \langle \psi | i \rangle \\ &= \langle \psi | A | \psi \rangle \end{aligned}$$

#### 2. Partial trace for quantum computing

In mixed quantum systems involving components  $A$  and  $B$ , described by a density matrix  $\rho^{AB}$ , the reduced density matrix for system  $A$  is defined as:



$$\rho^A \equiv \text{tr}_B(\rho^{AB}) = \text{tr}_B(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) \equiv |a_1\rangle\langle a_2| \text{tr}(|b_1\rangle\langle b_2|)$$

$\text{tr}_B$  represents the partial trace over system B.  $|a_1\rangle$  and  $|a_2\rangle$  are any two vectors in the state space of A, while  $|b_1\rangle$  and  $|b_2\rangle$  are any two vectors in the state space of B.

In fact, trace and partial trace are quantum operations and can be used to describe the outcome of measurement for a quantum circuit[22].

### 3.1.6 QUANTUM CIRCUIT

Similar to classical computer circuits, quantum computing also relies on circuits to process and manipulate information. However, quantum circuits fundamentally differ from their classical counterparts.

Quantum circuits consist of two essential operation components: quantum logic gates and measurements. These components serve as the foundation for quantum algorithms, giving them the potential to solve complex problems with greater efficiency and precision than classical computers.

#### QUANTUM GATE

Just as classical logic gates are used to change information, quantum logic gates serve a similar role but with distinct properties. Quantum logic gates facilitate quantum state transitions and manipulations of qubits, effectively transforming one quantum state into another, therefore, all the quantum gates are unitary quantum operators on a Hilbert space.

Quantum gates can be mathematically represented as unitary matrices of size  $2^n \times 2^n$ , where  $n$  is the number of qubits in a quantum system. For instance, consider the quantum NOT gate, analogous to the classical NOT gate,

$$\text{NOT}(\alpha|0\rangle + \beta|1\rangle) = \alpha|1\rangle + \beta|0\rangle. \quad (3.7)$$

It interchanges the roles of  $|0\rangle$  and  $|1\rangle$ , and this transformation can be expressed using a matrix, denoted as  $X$ ,

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; X \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}.$$

It is important to highlight that the sole constraint on a quantum gate is its unitarity, denoted as  $U^\dagger U = I$ , where  $U^\dagger$  represents the adjoint (conjugate transpose) of  $U$ . This unitarity constraint endows quantum gates with two crucial attributes: linearity and reversibility, thanks to the unique properties of the unitary matrix. Consequently, quantum gates can effectively maintain the probability of a quantum system's state. Moreover, this constraint underscores the reversibility of all operations on quantum bits, providing a robust foundation for quantum computation [22].

As for the properties of the quantum gate, according to the Pauli Decomposition, all 1-qubit unitary gates can be decomposed into rotations about the z-axis and the y-axis [22, 24], which represented by U3 gate in Qiskit and PennyLane.

$$U = e^{i\alpha} \begin{bmatrix} e^{-i\beta/2} & 0 \\ 0 & e^{i\beta/2} \end{bmatrix} \begin{bmatrix} \cos \frac{\gamma}{2} & -\sin \frac{\gamma}{2} \\ \sin \frac{\gamma}{2} & \cos \frac{\gamma}{2} \end{bmatrix} \begin{bmatrix} e^{-i\delta/2} & 0 \\ 0 & e^{i\delta/2} \end{bmatrix} = e^{i\alpha} R_z(\beta)R_y(\gamma)R_z(\delta)$$

Gate	Notation	Gate's qubit count	Matrix	Circuit
Pauli X gate	X, NOT, $\sigma_1, \sigma_x$	1	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	
Pauli Y gate	Y, $\sigma_2, \sigma_y$	1	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	
Pauli Z gate	Z, $\sigma_3, \sigma_z$	1	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	
X rotation gate	$R_x(\phi)$	1	$e^{-i\phi\sigma_x/2} = \begin{pmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{pmatrix}$	
Y rotation gate	$R_y(\phi)$	1	$e^{-i\phi\sigma_y/2} = \begin{pmatrix} \cos(\phi/2) & -\sin(\phi/2) \\ \sin(\phi/2) & \cos(\phi/2) \end{pmatrix}$	
Z rotation gate	$R_z(\phi)$	1	$e^{-i\phi\sigma_z/2} = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix}$	
Arbitrary single-qubit unitary gate	$U3(\theta, \phi, \lambda)$	1	$\begin{pmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda}\sin(\frac{\theta}{2}) \\ e^{i\phi}\sin(\frac{\theta}{2}) & e^{i(\phi+\lambda)}\cos(\frac{\theta}{2}) \end{pmatrix}$	
Hadamard	H	1	$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	
Controlled-NOT gate	CNOT; CX	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	
Controlled-Y gate	CY	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix}$	
Controlled-Z gate	CZ	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$	
Controlled-RX gate	CRX	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\frac{\theta}{2}) & 0 & -i\sin(\frac{\theta}{2}) \\ 0 & 0 & 1 & 0 \\ 0 & -i\sin(\frac{\theta}{2}) & 0 & \cos(\frac{\theta}{2}) \end{pmatrix}$	
Controlled-RY gate	CRY	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\frac{\theta}{2}) & 0 & -\sin(\frac{\theta}{2}) \\ 0 & 0 & 1 & 0 \\ 0 & \sin(\frac{\theta}{2}) & 0 & \cos(\frac{\theta}{2}) \end{pmatrix}$	
Controlled-RZ gate	CRZ	2	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$	
XX Rotation gate	RXX	2	$e^{-i\frac{\theta}{2}X\otimes X} = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 & 0 & -i\sin(\frac{\theta}{2}) \\ 0 & \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) & 0 \\ 0 & -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) & 0 \\ -i\sin(\frac{\theta}{2}) & 0 & 0 & \cos(\frac{\theta}{2}) \end{pmatrix}$	
YY Rotation gate	RYY	2	$e^{-i\frac{\theta}{2}Y\otimes Y} = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 & 0 & i\sin(\frac{\theta}{2}) \\ 0 & \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) & 0 \\ 0 & -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) & 0 \\ i\sin(\frac{\theta}{2}) & 0 & 0 & \cos(\frac{\theta}{2}) \end{pmatrix}$	
ZZ Rotation gate	RZZ	2	$e^{-i\frac{\theta}{2}Z\otimes Z} = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & e^{i\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & e^{i\frac{\theta}{2}} & 0 \\ 0 & 0 & 0 & e^{-i\frac{\theta}{2}} \end{pmatrix}$	

Table 3.1: Quantum gates (draw from PennyLane and Qiskit)

Besides, if we define the canonical gate as,

$$\begin{aligned} \text{CAN}(t_1, t_2, t_3) &= \text{RXX}(t_1\pi)\text{RYY}(t_2\pi)\text{RZZ}(t_3\pi) \\ &= e^{-i\frac{\pi}{2}t_1X\otimes X}e^{-i\frac{\pi}{2}t_2Y\otimes Y}e^{-i\frac{\pi}{2}t_3Z\otimes Z}. \end{aligned}$$

Then, following the Krauss and Cirac decomposition described in [24], any two-qubit gate can be decomposed into a canonical gate with two 1-qubit gates[25–27], see in the Figure 3.2.

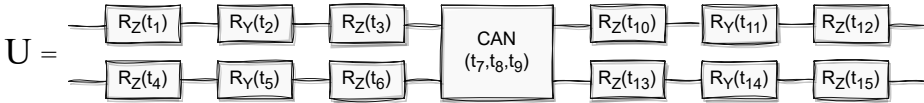


Figure 3.2: Random two-qubit gate decomposition.

It also can be written in this mathematical form,

$$U \equiv (A_1 \otimes A_2) \cdot e^{i(aX\otimes X + bY\otimes Y + cZ\otimes Z)} \cdot (A_3 \otimes A_4).$$

Here,  $a$ ,  $b$ , and  $c$  are real numbers, and  $A_1$ ,  $A_2$ ,  $A_3$ , and  $A_4$  represent single-qubit gates. It is noteworthy that such decompositions are universally applicable, irrespective of the specific forms of these single-qubit gates since they also have universal decomposition forms.

One crucial metric for evaluating quantum gate impact is the Entangling Power, which quantifies the extent to which various gates generate entanglement when applied to initially unentangled inputs. Notably, the CNOT gate is among those with the highest entangling power [28], underscoring its significance in quantum circuit design.

For the quantum gates employed in our work, as presented in Table 3.1.

### MEASUREMENT

In quantum circuits, the process of extracting data is known as measurement. There are two types of measurement outcomes have been used in our project:

1. Expectation value of an observable
  - In quantum mechanics, an observable, representing a property of an  $n$ -qubit system, is described by a quantum operator  $\mathcal{M}$  of dimensions  $2^n \times 2^n$
  - If the quantum state before measurement is denoted as  $|\psi\rangle$ , the expectation value of this observable  $\mathcal{M}$  is defined as:

$$\langle \mathcal{M} \rangle_\psi = \text{tr}(\mathcal{M}|\psi\rangle\langle\psi|) = \langle \psi | \mathcal{M} | \psi \rangle$$

2. Computational basis state probabilities
  - When measuring a qubit, we determine whether it's in the  $|0\rangle$  or  $|1\rangle$  state.
  - The outcome of multiple measurements results in probabilities associated with these computational basis states, which correspond to the amplitudes of the qubit  $\alpha$  and  $\beta$ .

Overall, the properties of quantum gates discussed above play a key role in determining how we design our quantum circuits.

### 3.1.7 QUANTUM GRADIENT

In quantum computing, the computation of gradients holds climactic significance, particularly in quantum machine learning-related quantum algorithms. In our paper, we employ two widely recognized methods for computing gradients.

Let's first consider a quantum model represented by the state  $|\psi(\theta)\rangle$ , where  $\theta$  denotes the model parameter vector. The circuit output, obtained using the observable  $M$ , is expressed as  $\langle\psi(\theta)|M|\psi(\theta)\rangle$ .

#### 1. Finite-Difference Method

The Finite-Difference Method is a numerical technique used to approximate derivatives by introducing small variations, denoted as  $\epsilon$ , to each parameter in a quantum circuit and observing the resulting changes. Rooted in the principle of finite differences, this method calculates the finite differences between the perturbed and unperturbed quantum circuit outputs.

In the context of the central finite difference method, the quantum gradient is approximated as follows,

$$\frac{\partial\langle\psi(\theta)|M|\psi(\theta)\rangle}{\partial\theta} \approx \frac{1}{2\epsilon} (\langle\psi(\theta+\epsilon)|M|\psi(\theta+\epsilon)\rangle - \langle\psi(\theta-\epsilon)|M|\psi(\theta-\epsilon)\rangle). \quad (3.8)$$

Theoretically, reducing  $\epsilon$  as much as possible brings the approximations closer to the true gradient. However, it is crucial to strike a balance, as excessively small  $\epsilon$  values can lead to numerical instability. Hence, selecting an appropriate  $\epsilon$  is essential for the reliability and efficiency of the Finite-Difference Method in quantum gradient computations.

#### 2. Parameter Shift Rule

In 2018, the parameter shift rule outlined by Mitarai et al. [29] started to join the field of quantum computing, marking a pivotal advance in quantum gradient computation. Since its introduction, this rule has developed into a sophisticated and highly efficient tool, providing a systematic methodology for obtaining precise gradients within quantum circuits.

At the very heart of the parameter shift rule is a comprehensive guide on how to utilize the same quantum circuit architecture to compute the partial derivatives of the circuit results. The mathematical expression is,

$$\frac{\partial\langle\psi(\theta)|M|\psi(\theta)\rangle}{\partial\theta} = c(\langle\psi(\theta+s)|M|\psi(\theta+s)\rangle - \langle\psi(\theta-s)|M|\psi(\theta-s)\rangle). \quad (3.9)$$

Here, the constants  $c$  and  $s$  are fully determined by the specific gates used in the quantum circuit [30]. This formula concisely captures the essence of the parameter shift rule, demonstrating its ability to facilitate gradient computation efficiently.

One notable feature of the parameter shift rule is its ability to leverage the same circuit for both computing the quantum function and its gradient. This characteristic significantly simplifies the computational complexity, reduces the complexity associated with separate computations, and marks a considerable advantage in quantum gradient computations.

### 3.1.8 FIDELITY

The concept of fidelity holds significant importance in the realm of quantum information [31], providing a mathematical framework for quantifying the similarity between two quantum states. Various fidelity measures exist, each taking different forms depending on the nature of the quantum states involved.

For a pair of arbitrary mixed quantum states, denoted as  $\rho$  and  $\sigma$ , the fidelity between them can be defined based on the closeness of these states in the standard geometry of Hilbert space. We can use Uhlmann-Jozsa (U-J) fidelity [32], denoted as  $\mathcal{F}(\rho, \sigma)$ , to describe it,

$$\mathcal{F}(\rho, \sigma) = (\text{tr} \sqrt{\sqrt{\rho}\sigma \sqrt{\rho}})^2. \quad (3.10)$$

When dealing with a pair of pure quantum states, expressed as  $\rho = |\phi\rangle\langle\phi|$  and  $\sigma = |\psi\rangle\langle\psi|$ . Then its fidelity is mathematically expressed as follows,

$$\mathcal{F}(\rho, \sigma) = \mathcal{F}(|\phi\rangle\langle\phi|, |\psi\rangle\langle\psi|) := |\langle\psi | \phi\rangle|^2. \quad (3.11)$$

Furthermore, when  $\rho$  represents a mixed state, this formulation can be extended to,

$$\mathcal{F}(\rho, \sigma) = \mathcal{F}(\rho, |\psi\rangle\langle\psi|) := \langle\psi|\rho|\psi\rangle. \quad (3.12)$$

This form generalizes fidelity as the transition probability between two quantum states. Essentially, it quantifies the likelihood of transitioning from one state to another, providing a probabilistic measure of their similarity. By studying the overlap of states in Hilbert space, fidelity captures the degree of consistency of quantum systems in their respective descriptions, providing valuable insight into their mutual similarity.

## 3.2 QUANTUM NEURAL NETWORK

As the problems solved with machine learning become more complex and the model sizes continue to grow, a significant challenge arises in the form of insufficient computational power in machine learning. Researchers are actively investigating ways to address this problem, and one possible direction is to integrate quantum computing with machine learning. This integration aims to enhance the efficiency of processing classical data and has emerged as a hot research topic in recent years.

Leveraging the considerable success of neural networks, there has been a strong interest in exploring the potential advantages between quantum computing and neural networks, leading to the concept of quantum neural networks. Some innovative ideas have been proposed by researchers, each showcasing varying degrees of similarity to classical neural networks. In this section, we will introduce two types of quantum neural networks which are the basis of our study. The models under consideration, namely dissipative quantum neural networks and parameterized variational quantum neural networks, will be presented and thoroughly discussed in the following part.

### 3.2.1 DISSIPATIVE QUANTUM NEURAL NETWORK

The Dissipative Quantum Neural Network (DQNN) was initially introduced by Beer [12] and has been applied by several researchers in the field of finance [19, 20]. This quantum neural

network shares a lot of similarities with classical neural networks, as both frameworks consist of multiple perceptrons to process information. Furthermore, they employ back-propagation to iteratively update model parameters.

### QUANTUM PERCEPTRONS

Quantum perceptrons within this framework are prepared as arbitrary quantum gates, each accompanied by an ancilla qubit. A Quantum perceptron operates on a total of  $m + 1$  qubits, where  $m$  qubits are from the preceding layer or input, denoted as  $\rho^{in}$ , and the additional qubit is the ancilla qubit initialized in the state  $|0\rangle$ . The perceptron operation involves applying the quantum gate, and subsequently, partial tracing over the  $m$  input qubits, resulting in an outcome state of that single ancilla qubit, denoted as  $\rho^{out}$ . For a visual illustration, refer to the example presented in Figure 3.3.

3

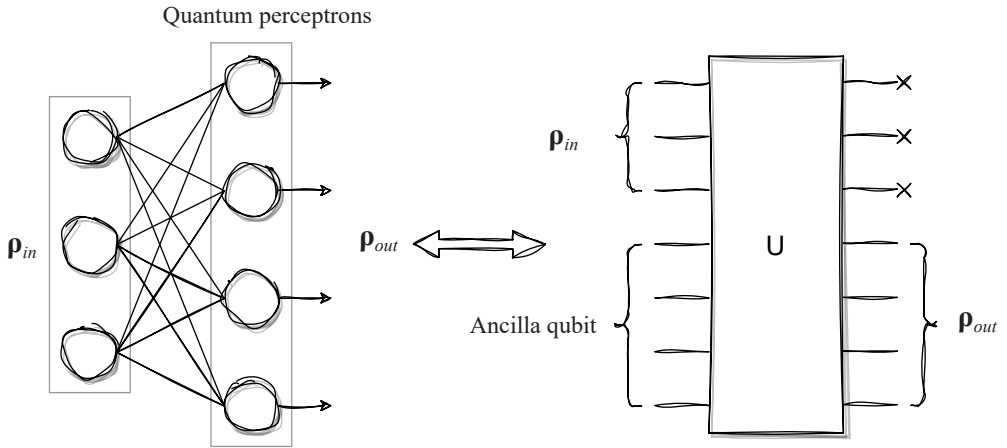


Figure 3.3: Quantum perceptron.

### THE STRUCTURE OF DISSIPATIVE QUANTUM NEURAL NETWORK

Similar to classical neural networks, DQNN also consists of three types of layers: one input layer, one output layer, and  $L$  hidden layers between the input layer and the output layer, as depicted in Figure 3.4. The input layer consists only of qubits, while the hidden and output layers are constructed by quantum perceptrons.

### FEED-FORWARD AND BACK-PROPAGATION

For data processing and network optimization, DQNN utilizes similar feed-forward and back-propagation schemes that are tightly integrated with classical neural networks. These parallel strategies facilitate the seamless flow of data through the network, leading to efficient processing and subsequent fine-tuning of the entire system. In the next sections, a comprehensive exploration of the intrinsic complexity of feed-forward and back-propagation will clarify their key role in the DQNN framework. The integration of classical methods with quantum networks highlights the versatility and adaptability of DQNN in dealing with complex information-processing tasks.

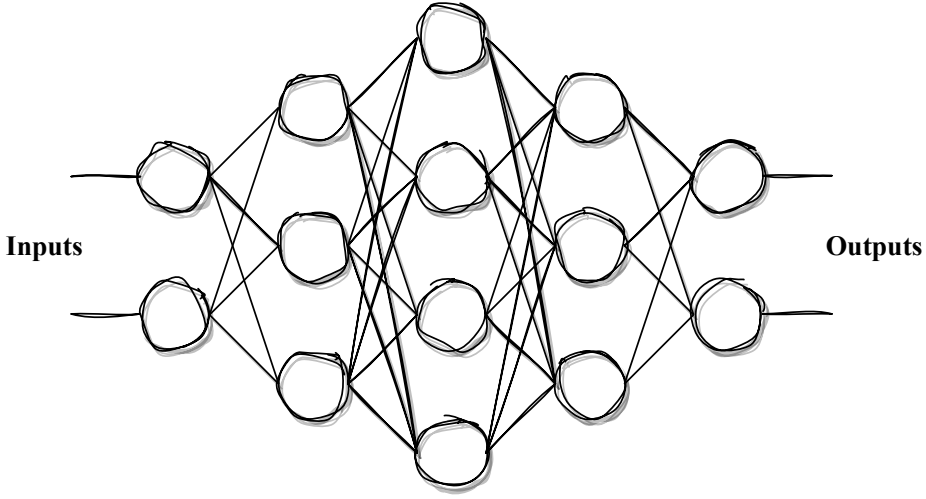


Figure 3.4: The structure of a 2-3-4-3-2 Dissipative Quantum Neural Network.

### 1. Feed-forward

For each layer within the DQNN, it can be seen as a quantum operation that transforms the output from the preceding layer into the input for the subsequent layer. Notably distinct from classical networks, the implementation of quantum perceptron units in each layer follows a sequential order, from the top perceptron to the bottom perceptron.

Therefore, by the definition from Equation (3.5), we have

$$\rho^l = \mathcal{E}^l(\rho^{l-1}) \equiv \text{tr}_{l-1} \left( \prod_{j=m_l}^1 U_j^l (\rho^{l-1} \otimes |0 \dots 0\rangle_{l-1} \langle 0 \dots 0|) \prod_{j=1}^{m_l} U_j^{l\dagger} \right). \quad (3.13)$$

The quantum gate corresponding to the  $j^{\text{th}}$  perceptron in the  $l^{\text{th}}$  layer is denoted as  $U_j^l$ , where  $j = 1, \dots, m_l$  and  $l = 1, \dots, L, L+1$ , with  $l = 1$  representing the first hidden layer and  $l = L+1$  denoting the output layer.

The output from the  $l^{\text{th}}$  layer during the feed forward process, designated as  $\rho^{l+1}$ , with initial state  $\rho^0 = \rho^{\text{in}}$  and final state  $\rho^{L+1} = \rho^{\text{out}}$ , contains the joint outcomes of all  $m_l$  perceptrons within that layer. Moreover, considering the entire network, we can succinctly formulate the output state of the DQNN  $\rho^{\text{out}}$  using Equation (3.14).

$$\rho^{\text{out}} = \mathcal{E}(\rho^{\text{in}}) = \mathcal{E}^{L+1}(\mathcal{E}^L(\dots \mathcal{E}^2(\mathcal{E}^1(\rho^{\text{in}})) \dots)) \quad (3.14)$$

### 2. Back-propagation

The adjoint form of Equation (3.13), which can be derived by using Kraus decomposition, plays a crucial role in representing the quantum operation during DQNN back-propagation, and it can be expressed as,

$$\sigma^{l-1} = \mathbb{P}^l(\sigma^l) = \text{tr}_l((\mathbb{1}_{l-1} \otimes |0 \dots 0\rangle_l \langle 0 \dots 0|_l) \prod_{j=1}^{m_l} U^{l\dagger} (\mathbb{1}_{l-1} \otimes \sigma^l) \prod_{j=m_l}^1 U_j^l). \quad (3.15)$$

This equation represents the quantum state evolution throughout the back-propagation process. During backpropagation, the quantum state labeled as  $\rho^{\text{label}}$  initially serves as the input to the output layer of the DQNN. Subsequently,  $\rho^{\text{label}}$  is propagated backward through the layers until it reaches the input layer of the network. The outcomes at each layer during the backpropagation process can be computed using Equation (3.15).

3

### TRAINING PROCESS

Now we train a DQNN with  $N$  training data pairs, each of which is structured as  $[|\phi_i^{\text{in}}\rangle, |\phi_i^{\text{label}}\rangle]$ ,  $i=1 \dots N$ . The entire training process is shown in the Figure 3.5. The initial step involves the random initialization of all quantum gates within the quantum perceptrons. Subsequently, the feed-forward and back-propagation processes are performed simultaneously. For each

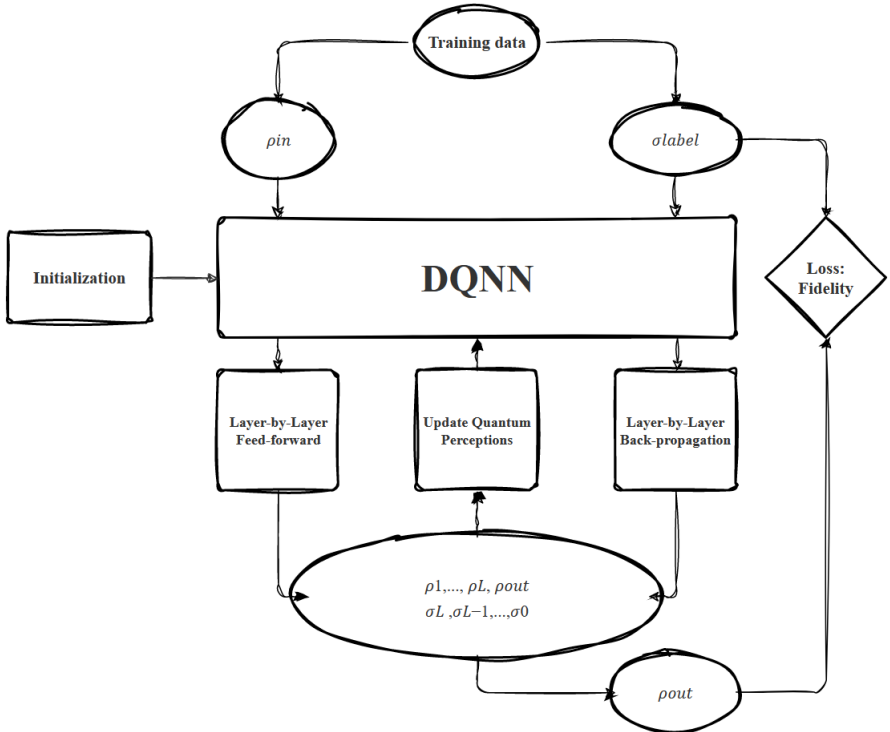


Figure 3.5: Training process of DQNN

training data pair,  $\rho_i^{\text{in}} = |\phi_i^{\text{in}}\rangle\langle\phi_i^{\text{in}}|$ ,  $i=1, \dots, N$ , serves as the input to the first hidden layer for the feed-forward. Afterward, the feed-forward transformations are sequentially applied



layer by layer until reaching the output layer. The resulting quantum states denoted  $\rho_i^1, \rho_i^2, \dots, \rho_i^L$  and  $\rho_i^{L+1}$  (corresponding to  $\rho_i^{out}$ ), are obtained through this process and can be calculated by Equation (3.13). Concurrently,  $\sigma_i^{label} = |\phi_i^{label}\rangle\langle\phi_i^{label}|$  (also seen as  $\sigma_i^{L+1}$ ) is used as the input for the output layer, initiating the back-propagation transformation from the output layer to the first hidden layer. The outcomes,  $\sigma_i^L, \sigma_i^{L-1}, \dots, \sigma_i^0$ , are preserved according to Equation (3.15).

Next, the obtained quantum states  $\rho_i^{out}$  and label quantum states  $\sigma_i^{label}$  can be used to compute the training loss for our DQNN model. For the loss function, we use the mean fidelity value between  $\rho_i^{out}$  and  $\sigma_i^{label}$ ,  $i=1, \dots, N$ , providing a measure of how well the output state aligns with the labeled target state. The fidelity, denoted as  $\mathcal{F}(\rho_i^{out}, \sigma_i^{label})$ , yields a value of 1 when the output and label are identical.

Given that  $\rho_i^{out}$  is a mixed quantum state and  $\sigma_i^{label}$  is a pure quantum state, we formulate the loss using the fidelity definition from Equation (3.12) as follows,

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N F(\sigma_i^{label}, \rho_i^{out}) = \frac{1}{N} \sum_{i=1}^N F\left(|\phi_i^{label}\rangle\langle\phi_i^{label}|, \rho_i^{out}\right) = \frac{1}{N} \sum_{x=1}^N \langle\phi_x^{label} | \rho_x^{out} | \phi_x^{label}\rangle. \quad (3.16)$$

Furthermore, the update process involves modifying the quantum perceptrons based on the computed gradients and the collected results from the feed-forward and back-propagation phases, see the details in [12]. Equation (3.17) captures this evolution, where the quantum states at each layer,  $\rho_i^0$  (which is  $\rho_i^{in}$ ),  $\rho_i^1, \rho_i^2, \dots, \rho_i^L, \sigma_i^{L+1}, \sigma_i^L, \dots, \sigma_i^1$  are utilized to adjust the respective unitary operators  $U_j^l$ .

$$e^{i\epsilon K_j^l} U_j^l \rightarrow U_j^l(\text{new}), \quad (3.17)$$

where

$$K_j^l = \frac{\eta 2^{m_l-1} i}{N} \sum_{x=1}^N \text{tr}_{rest}(M_j^l(i)),$$

and

$$M_j^l(i) = [U_j^l \dots U_1^l (\rho_i^{l-1} \otimes |0\dots 0\rangle\langle 0\dots 0|) U_1^{l\dagger} \dots U_j^{l\dagger}, U_{j+1}^l \dots U_{m_l}^l (\mathbb{1}_{in, hid} \otimes \sigma_i^l) U_{m_l}^l \dots U_{j+1}^l].$$

$\epsilon$  is an adjustable parameter and the parameter matrix  $K_j^l$  is instrumental in this adjustment and is determined by the trace operation on the remaining degrees of freedom, all the qubits that are not affected by  $U_j^l$ , and the learning rate  $\eta$ , as quantified by  $M_j^l(i)$ .

The process of updating is iterated until the obtained loss function converges to 1. Essentially, this update mechanism ensures that the quantum neural network adapts its internal representations, which are enclosed in the quantum perceptron, to optimize overall performance based on the feedback received during training.

### 3.2.2 PARAMETERIZED VARIATIONAL QUANTUM NEURAL NETWORK

The Parameterized Variational Quantum Neural Network (PVQNN), also known as Variational Quantum Circuits (VQC), Quantum Circuit Learning (QCL) [29], and Parameterized Quantum Circuits (PQCs) [13], represents a powerful quantum-classical hybrid model.

This model consists of a quantum circuit containing trainable parameters connected with a classical optimizer. Usually, PVQNN is regarded as a quantum neural network model, since emphasizes the modular structure of quantum gates within a circuit, drawing parallels with classical neural networks. Besides, the utilization of optimization techniques derived from classical neural network training is also a notable characteristic of PVQNN [33].

PVQNN serves as a versatile framework with applications in solving classical problems in finance, chemistry simulations, and other optimization tasks. Research and developments in PVQNN are actively pursued by well-known organizations, including PennyLane, IBM, and various research institutions. This widespread adoption emphasizes the significance and potential of PVQNN in advancing quantum computing applications.

3

### THE STRUCTURE OF PARAMETERIZED VARIATIONAL QUANTUM NEURAL NETWORK

PVQNNs typically initiate their computation with several qubits prepared in the computational basis state, often  $|0\rangle$ , and undergo a sequence of operations before being measured at the end. The PVQNN architecture comprises three distinctive layers: the Data Encoding Layer (or Feature Mapping Layer), the Hidden Layer (Data Processing Layer), and the Measurement Layer, as illustrated in Figure 3.6.

Therefore, the output of the PVQNN can be described by the expression,

$$f_{\theta}(x) = \langle 0|U_e^{\dagger}(x)U_h^{\dagger}(\theta)MU_h(\theta)U_e(x)|0\rangle. \quad (3.18)$$

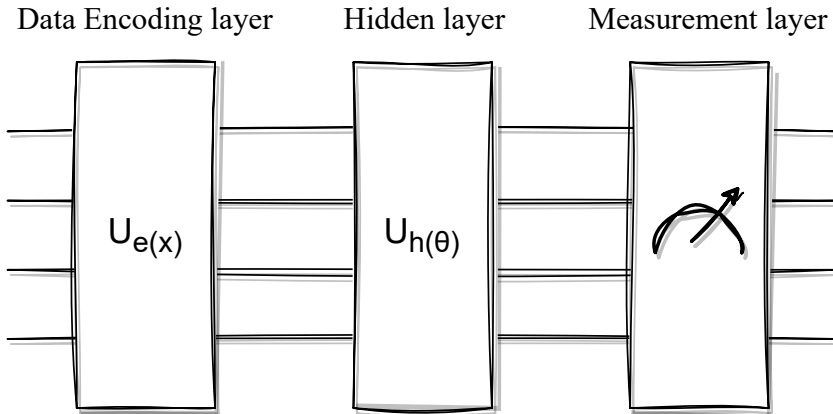


Figure 3.6: The structure of PVQNN.

For detailed information on each layer, see below,

#### 1. Data Encoding Layer

The data encoding layer serves as the initial stage, responsible for embedding classical information  $x$  into the quantum circuits using diverse data encoding methods. This layer is crucial for preparing the classical data for subsequent quantum processing. Mathematically, this process can be briefly described by a unitary matrix  $U_e(x)$ .

## 2. Hidden Layer

The hidden layer acts as the core data processing unit within the PVQNN. Each hidden layer can be viewed as a "block" formed of quantum gates. These gates include quantum gates with variable parameters  $\theta$  (e.g., rotation gates) and other gates (e.g., gates that increase the degree of entanglement like CNOT gates). These parameters are initially unknown but are continuously updated during training. The repetition of the parameterization layer enhances the expressiveness of the model and allows for complex quantum operations. The operations in this layer are mathematically represented by a unitary matrix  $U_h(\theta)$ .

## 3. Measurement Layer

The measurement layer in a PVQNN serves the crucial function of extracting classical information from the quantum states resulting from the previous layers' computations. This layer plays a pivotal role in subsequent optimization procedures. Specifically, the outcomes of the measurements are usually obtained by taking the expectation values of certain chosen observables, commonly involving the subset of Pauli gates, namely I (Identity), X, Y, Z.

### FOURIER PERSPECTIVE

To enhance comprehension of the PVQNN model, researchers have explored it from various perspectives. In this section, we provide an in-depth understanding of PVQNN through the lens of the Fourier series.

When considering the data encoding layer as a transition of quantum states in time, it can be represented by a unitary operator  $U_e(X) = e^{-iXH}$ ,  $X \in \mathbb{R}^N$ , also known as a time evolution operator. Here,  $H$  is a time-independent Hamiltonian operator [34], and

$$U_e(X) := e^{-ix_1 H_1} \otimes \dots \otimes e^{-ix_N H_N}$$

For the Hamiltonian operator  $H$ , which is a Hamiltonian matrix, the finite-dimensional spectral theorem allows us to express  $H = V^\dagger \Sigma V$  through Schur decomposition, where  $V$  is a unitary matrix and the eigenvalues of  $H$  are the diagonal entries of diagonal matrix  $\Sigma$ .

Since  $V$  is a unitary matrix, transformations  $|\Gamma\rangle = V|0\rangle$  and  $\tilde{U}_h = U_h V^\dagger$  preserve the unitary property of  $M$  and  $U_h$ . Consequently, Equation (3.18) can be reformulated as,

$$\begin{aligned} f_{\theta}(X) &= \langle 0 | e^{iXH^\dagger} U_h^\dagger(\theta) M U_h(\theta) e^{-iXH} | 0 \rangle \\ &= \langle 0 | V^\dagger e^{iX\Sigma} V U_h^\dagger(\theta) M U_h(\theta) V^\dagger e^{-iX\Sigma} V | 0 \rangle \\ &= \langle \Gamma | e^{iX\Sigma} \tilde{U}_h^\dagger M \tilde{U}_h e^{-iX\Sigma} | \Gamma \rangle. \end{aligned} \quad (3.19)$$

If we denote the eigenvalues of the Hamiltonian  $H_k$  as  $\lambda_1^k, \dots, \lambda_d^k$ ,  $d$  is the dimension of the Hilbert space and  $k \in 1, \dots, N$ . Then,

$$(e^{-ix_k \Sigma_k})_{j,j} = e^{-ix_k \lambda_j^k}$$

Since the presence of  $e^{iX\Sigma}$  in the Equation (3.19), so there are factors  $e^{ix_k \lambda_s^k} e^{-ix_k \lambda_t^k} = e^{-ix_k (\lambda_s^k - \lambda_t^k)}$  in the  $f_{\theta}(X)$ . Now, let's define the frequency spectrum  $\Omega_i$  as,

$$\Omega_i = \lambda_s^k - \lambda_t^k,$$

where  $s, t \in 1, \dots, d$ . This leads us to a more compact expression,

$$f_{\theta}(x) = \sum_{\omega_1 \in \Omega_1} \dots \sum_{\omega_N \in \Omega_N} C(\theta) e^{i\omega_1 x_1} \dots e^{i\omega_N x_N}. \quad (3.20)$$

Here, the coefficients  $C(\theta)$  incorporates terms arising from  $\tilde{U}_h$  and measurement layer  $M$ , see [35] for more details.

Notably, in Equation (3.20), considering that  $\lambda_s$  and  $\lambda_t$  are any two eigenvalues from the same Hamiltonian matrix, it leads to two key conclusions: firstly, 0 is an element of  $\Omega$ , and secondly, there exists a symmetry property whereby  $-\omega$  is in  $\Omega$  whenever  $\omega$  is. These factors collectively suggest that  $f_{\theta}(x)$  can appropriately be conceptualized as a multi-dimensional Fourier series.

From above, the frequencies  $\omega$ , as delineated in  $f_{\theta}(x)$ , exhibit a direct correlation with the spectra of the encoding Hamiltonians  $H$ . These Hamiltonians, integral to the encoding layer, are crucial in determining the specific frequencies that characterize the behavior of the quantum neural network. On the other hand, the Fourier coefficients  $C$  are mainly shaped by the network's overall architecture and the design decisions involved in its construction. This encompasses factors such as the configuration of the encoding layer, the structural design of the hidden layers, and the selection of observables.

In essence, the encoding Hamiltonians determine the fundamental frequencies. Meanwhile, the network's architecture, including the design of the encoding and hidden layers, as well as the choice of observables, collectively influences the Fourier coefficients in Equation (3.20). This correlation between the network's structural design and its Fourier series representation allows researchers to adeptly customize the quantum neural network, enhancing its sensitivity and expressibility to specific data features or targeted tasks.

### TRAINING PROCESS

The training process of a PVQNN model shares many similarities with classical neural networks, particularly because it operates as a hybrid model. When training a PVQNN with  $N$  classical training data pairs  $[x_i, y_i]$ ,  $i = 1, \dots, N$ , the overall process, as illustrated in Figure 3.7, can be outlined as follows,

#### 1. Initialization

The training begins by preparing qubits in the computational basis state  $|0\rangle$ . The initialization phase also involves critical decisions like selecting the data encoding method and the way the hidden layers are constructed, such as the selection and arrangement of quantum gates contained within them. These quantum gates are initially set with random values, similar to the initialization of weights in a classical neural network. An observable for the measurement layer is also chosen, which is key to translating the quantum computations into classical outputs.

#### 2. Data Encoding

This step involves converting classical input data into a quantum format that the PVQNN can process. It's done by applying a set of unitary gates  $U_e(x_i)$  to the initialized qubits. This transformation encodes the classical data  $x_i$  into a quantum state  $|\phi_{in}(x_i)\rangle = U_e(x_i)|0\rangle$ , effectively bridging the gap between classical and quantum information.

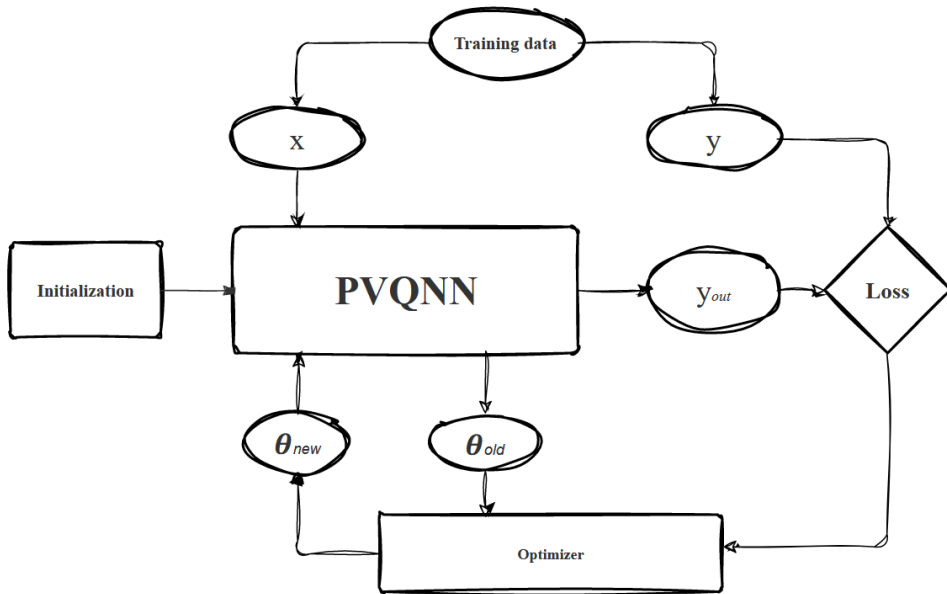


Figure 3.7: Training process of PVQNN.

### 3. Data Processing

This step passes the encoded qubits obtained from the data coding layer through the hidden layer. This involves applying parameterized unitary gates  $U_h(\theta)$  to the input quantum state  $|\phi_{in}(x_i)\rangle$  and generate an output state  $|\phi_{out}(x_i)\rangle = U_h(\theta)|\phi_{in}(x_i)\rangle$ . These gates manipulate the quantum state in a way that's determined by the network's current parameters  $\theta$ , thus performing the quantum equivalent of data processing.

### 4. Measurement

After processing, the quantum state is measured. This step is crucial in quantum computing as it converts the quantum information back into a classical form. The measurement is performed using predefined observables  $M$ , and the outcome  $y_i^{out} = \langle \phi_{out}(x_i) | M | \phi_{out}(x_i) \rangle$  provides the classical data needed for further steps. This is where the quantum computation results become accessible for classical analysis.

### 5. Optimization

The final step reproduces the optimization process in classical neural networks. It involves selecting a loss function  $L(y_i^{out}, y_i)$  and a classical optimizer. The loss function measures the difference between the network's output and the actual desired output. In the case of a gradient-based optimizer, a unique aspect is the computation of quantum gradients, which presents both challenges and advantages compared to classical methods. The network is iteratively improved through this optimization process, refining the parameters to reduce the loss function, thereby enhancing the network's performance.

This process leverages the principles of quantum mechanics, potentially offering computational advantages over classical approaches in certain scenarios.

# 4

## QUANTUM NEURAL NETWORK TO COMPUTE IMPLIED VOLATILITY

4

*In this chapter, we introduce the algorithmic framework and the experimental setup for our work. Section 4.1 outlines the model's general procedure, customized for our specific task. In Section 4.2, we explore the comprehensive configuration of our selected models. This includes everything from data encoding to the detailed settings of each model type. We pay special attention to the setting of PVQNN, including the data re-uploading technique, the architecture of the quantum circuit, the choice of observables, and the key optimization parameters essential for the effective training of the model.*

## 4.1 ALGORITHM

In our study, we introduce two different quantum neural network models, DQNN and PVQNN, designed to approximate the inverse of the Black-Scholes formula,  $\sigma = BS^{-1}(V, K, \tau, S, r)$ , for calculating implied volatility. These models utilize the unique computational capabilities of quantum computing, aiming to investigate the distinct features and potential benefits of quantum neural networks in addressing this problem.

The process we adopt for these quantum neural network models is detailed in a structured format in Table 4.1. This table comprehensively delineates each step of the process, which is broadly categorized into two primary phases: data preparation and model training & evaluation. The structured layout provides a clear and methodical guide for the implementation and assessment of the DQNN and PVQNN models in related fields.

Table 4.1: Quantum Neural Network Model Framework

Step	Description
1	<b>Data Generation and Dataset Division:</b> To generate training data, instead of generating training data through root-finding algorithm $\sigma = BS^{-1}(V^{mkt}, K, \tau, S, r)$ , we initially sample values for $S, K, \tau, r, \sigma$ and then compute the option price $V$ using the Black-Scholes formula $V = BS(S, K, \tau, r, \sigma)$ . The input $x = (S, K, \tau, r, V)$ is fed into the quantum neural network with sampled data $\sigma$ serving as the label data $y = \sigma$ . Subsequently, we divide the complete dataset into two subsets: one for training and the other for testing the performance of the quantum neural network.
2	<b>Data Encoding:</b> Transform the classical dataset into a quantum-computable format, normally the state of qubits, using various data encoding techniques. In the case of PVQNN, this conversion is performed within the Data Encoding Layer.
3	<b>Model Training:</b> Continuously train the quantum neural network using the training dataset, focusing on minimizing the loss function $L$ until it converges to a satisfied value.
4	<b>Model Evaluation:</b> Once the training is complete and the loss function is optimized, assess the quantum neural network's performance using the testing dataset.

## 4.2 MODEL SETUP

Just like their classical counterparts, the configuration and settings of quantum neural networks are key in determining their performance. In this section, we delve into the specific settings of the DQNN and PVQNN models based on their characteristics.

We will examine key aspects such as the architecture of the quantum circuits, the choice and arrangement of quantum gates, and the methods of data encoding and decoding specific to quantum computing. The role of hyperparameters, including the number of qubits and layers, and their optimization strategies will also be explored.



Furthermore, we aim to investigate how these settings affect the models' ability to handle financial data, particularly in terms of scalability, accuracy, and computational cost. By understanding and fine-tuning these parameters, we can enhance the QNNs' capacity to model complex financial instruments and predict market dynamics, paving the way for more advanced applications in quantum finance. This exploration will not only provide deeper insights into the operational complexities of DQNN and PVQNN but also guide future developments in quantum neural network design and application.

### 4.2.1 DATA ENCODING

To effectively implement a quantum machine learning algorithm for solving classical problems, a primary challenge is the conversion of classical data into quantum information that quantum devices can process. Typically, this is achieved by embedding classical data into the quantum states of qubits through various data encoding methods. These methods, outlined below, are essential for ensuring that the quantum algorithm can interpret and manipulate the classical data effectively. It not only bridges the gap between classical and quantum computing but also leverages the unique properties of quantum systems to enhance computational capabilities.

#### 1. Basis encoding

Basis encoding, a straightforward and intuitive method, involves the transformation of binary data  $x$  of length  $n$  into a quantum state  $|\psi\rangle = |x_i\rangle^{\otimes n}$  with  $n$  qubits. Here, each  $|x_i\rangle$  represents a quantum computational basis state, existing either as  $|0\rangle$  or  $|1\rangle$ . For instance, consider a classical data  $x = 2$  with its binary equivalent being 10. In this case, the corresponding quantum state is  $|01\rangle$ .

This method is particularly well-suited for datasets comprised of small integers. This suitability arises from the limitations of current quantum computing devices, which are only capable of supporting quantum networks of a certain size. Encoding large or decimal numbers would require a significantly higher number of qubits, making the process more complex and resource-intensive. Additionally, this method benefits from the straightforward relationship between quantum bit representation and classical binary representation. This direct correlation allows for a more efficient and less complicated conversion of classical data into quantum information.

#### 2. Amplitude encoding

Amplitude encoding encodes classical data, denoted as  $x \in \mathbf{R}^N$ , into a quantum state  $|\psi\rangle$  using  $n$  qubits, where  $n = \log_2(N)$ . This process involves normalizing the classical data through a function  $f_i$ , resulting in the quantum state,

$$|\psi\rangle = \sum_{i=1}^N f_i(x)|i\rangle.$$

In this expression,  $\sum_i |f_i|^2 = 1$  ensures that the sum of the squares of the amplitudes equals one, following the principles of quantum mechanics. Each  $|i\rangle$  represents a computational basis in the Hilbert space, such as  $|01\rangle$ ,  $|11\rangle$ , etc.

Amplitude encoding allows the representation of an exponentially large amount of classical data with a relatively small number of qubits. This method is particularly useful for complex datasets or when dealing with high-dimensional data, as it can encode a large volume of information in a compact quantum form. However, the requirement for normalization and the maintenance of the quantum state's validity can pose challenges, particularly in ensuring that the transformed data accurately reflects the features of the original dataset. Moreover, amplitude encoding can be highly sensitive to errors, as small changes in amplitude can significantly alter the encoded information. This sensitivity necessitates precise control and error correction mechanisms in quantum algorithms employing this encoding technique. Besides, amplitude coding cannot simply be represented as a circuit form. In fact, the implementation of amplitude coding requires using a method called arbitrary state preparation which requires a large number of quantum gates [36].

### 3. Time-evolution Encoding

Time-evolution encoding generally refers to an encoding method represented by the evolution time of a specific Hamiltonian,  $\hat{H}$ . This process can be mathematically expressed as,

$$\phi(x) = e^{-i\hat{H}x}|0\rangle.$$

In this method, the initial quantum state  $|0\rangle$  is evolved through the action of the Hamiltonian operator, which is modulated by the classical data  $x$ . This evolution effectively encodes the classical data into the quantum state.

A commonly employed variant of Time-evolution Encoding is angle encoding, which utilizes rotation gates to encode the classical data  $x$ . In this approach, the angles through which these gates rotate are directly determined by the classical data. In our study, we primarily focus on angle encoding due to its ease of implementation in quantum circuits and its compatibility with the characteristics of our dataset. The specifics of these features are elaborated in Section 5.1.

#### 4.2.2 DQNN

In our implementation of the DQNN, we considered the time-intensive nature of training such networks on classical simulators. Drawing from Beer et al.'s research [25], we noted that a training session for a 2-3-4-3-2 DQNN architecture with 100 data points over 300 rounds requires approximately 80 minutes for simulation. Besides, the size of the DQNN significantly influences the duration of the training process. Hence, we selected a 2-3-4-3-2 structure DQNN to approximate the function.

Our implementation process begins with encoding the input features  $x$  (suppose we have four features) into two qubits. These qubits are initially in the  $|0\rangle$  state and are encoded using the Dense Angle Encoding method  $e_4$ , see the detail in Section 5.3.2, resulting in the input quantum state  $\phi^{in}$ . The data then passes through the network. Specifically, the first hidden layer contains three quantum perceptrons, followed by a second hidden layer with four quantum perceptrons, and a third hidden layer with three quantum perceptrons. The

network ends in the output layer, comprising two quantum perceptrons, which yield the output quantum state  $\rho^{out}$ .

The label training data  $y$ , containing the implied volatility  $\sigma$ , is encoded into a quantum state  $\phi^{lable}$  using a single Ry rotation gate, represented by  $e_2$ , see Section 5.3.2. This encoded state is then entangled with a qubit in the  $|0\rangle$  state to match the size of the output state  $\rho^{out}$ . This entanglement is critical for computing the fidelity between the predicted and actual outputs, a key step in the training process. Based on the label training data, we iteratively update the perceptrons in both the hidden and output layers. For this process, we set the learning rate  $\lambda$  to 1, and the adjustable parameter ( $\epsilon$ ) is established at 0.1, aligning with the parameters used in Sakuma's study [19]. The total number of iterations for this update process is fixed at 1000, allowing the network sufficient time to converge, as detailed in Table 4.2.

Table 4.2: The setting of DQNN.

Setting	Option
Structure	2-3-4-3-2
Data size	training:160 testing:40
Data encoding method	$\phi^{lable}: e_2$ $\phi^{in}: e_4$
Loss function	Fidelity
Learning rate $\lambda$	1
Adjustable parameter $\epsilon$	0.1
Iterations number	1000

### 4.2.3 PVQNN

As the PVQNN is currently in its developmental stages, established norms and best practices have yet to be solidified, presenting both challenges and opportunities for innovation.

In the data encoding layer of the PVQNN, there is no requirement for ancilla qubits in its structure. As a result, each input feature can be efficiently encoded into a single qubit. This direct encoding means that the total number of qubits,  $N$ , matches the number of input features.

Besides, the efficacy of the PVQNN is significantly influenced by its structural design and the optimization of various factors. This section aims to methodically study these elements to better understand their impact, particularly in their application to implied volatility prediction.

Initially, we focus on the design of the ansatz structure. Central to this is the data re-uploading technique, a method important to the model's operation. Subsequently, the selection of observables for measurement is explored. We consider various observables and their influence on the parameters' gradients.

Finally, we discuss the hyperparameters in the classical optimizer and the settings in the training. The choice and tuning of these hyperparameters are pivotal in optimizing the model's learning process and ultimately its predictive performance. By comparing different optimization strategies and training settings, we aim to identify configurations that enhance the PVQNN's efficiency and accuracy for implied volatility prediction.

This comprehensive examination is designed to clarify the pivotal factors that influence the PVQNN's performance, providing some insight for future research and development in similar applications.

### DATA RE-UPLOADING

The data re-uploading technique, as highlighted in the work of Pérez-Salinas et al. [37], plays a crucial role in the functionality of the model. This method involves encoding classical information multiple times, enabling the model to adaptively process and integrate data, just like the Residual network (ResNet). It appears to offer a solution to the challenges posed by the no-clone rule in quantum computing. In classical neural networks, the same input is processed repeatedly to the neurons in the later hidden layer. However, in PVQNN, each input data can be used only once. Therefore, re-uploading data becomes a practical approach to achieve the same effect within the quantum circuit framework.

For PVQNN, data re-uploading introduces input redundancy, which can be used to enhance model performance, as suggested by Vidal and Theis [38]. This technique has also been widely adopted in related studies [39–42]. Consequently, the utility and effectiveness of data re-uploading in improving the model's capability to manage complex financial datasets deserve a thorough examination.

There are two primary methods for data re-uploading for PVQNN: within the encoding layer and the hidden layers. When data re-uploading occurs in the encoding layer, the PVQNN can be described by the following equation,

$$f_{\theta}(x) = \langle 0 | U_e^{\dagger} \dots U_e^{\dagger}(x) U_h^{\dagger}(\theta) M U_h(\theta) U_e(x) \dots U_e(x) | 0 \rangle. \quad (4.1)$$

In this approach, the data is repeatedly encoded at the beginning, processing the information before it is passed through the hidden layers.

Alternatively, data re-uploading within the hidden layers involves starting each hidden layer with an encoding layer. In this scenario, the PVQNN is represented as,

$$f_{\theta}(x) = \langle 0 | U_e^{\dagger} U_h^{\dagger}(\theta) \dots U_e^{\dagger}(x) U_h^{\dagger}(\theta) M U_h(\theta) U_e(x) \dots U_h(\theta) U_e(x) | 0 \rangle. \quad (4.2)$$

Here, the data is re-uploaded at the beginning of each hidden layer. The frequency of data re-uploading corresponds to the number of hidden layers in the model. In our study, we will focus primarily on this second method of re-uploading. This decision is informed by the observation that some encoding strategies, such as simple angle rotation, may not be effective in the first approach. Specifically, rotating the data multiple times within the Encoding Layer might not significantly alter the outcomes, thus making the second method more suitable for our analysis.

### ANSATZ STRUCTURE

The structure of the hidden layer, also known as the ansatz, is intrinsically related to how the model processes data. Viewing each layer of the hidden layer as a modular component, the entire hidden layer is constructed using these modules.

The construction of these modules typically involves two key components: parameterized gates and gates for entanglement. In designing the ansatz, we aim for it to represent several critical features:

### 1. Universality

It is essential for the gates to have the capability to represent any arbitrary quantum gates. This universality is crucial in enhancing the expressibility of the model, allowing it to effectively capture and represent a wide range of quantum states and transformations.

### 2. Trainability

The modules must include parameterized gates to facilitate subsequent training. To optimize for efficiency in gradient-based training methods, the total number of parameters in the hidden layer should be kept below 100, as a higher parameter number can significantly increase training time.

## OBSERVABLES

Once the data encoding methods and ansatz structure for the quantum circuits have been established, the remaining variable for circuits to consider is the choice of observables. For a four qubits quantum circuit, and is designed to produce a single output, the potential choices for observables amount to 256, which is because the observable for each qubit can be selected from four different single-qubit Pauli operations:  $\sigma_x, \sigma_y, \sigma_z, I$ . Our empirical investigations have shown that varying the type of observables significantly impacts the gradient values. This finding is critical as it can influence the efficiency and effectiveness of the training process.

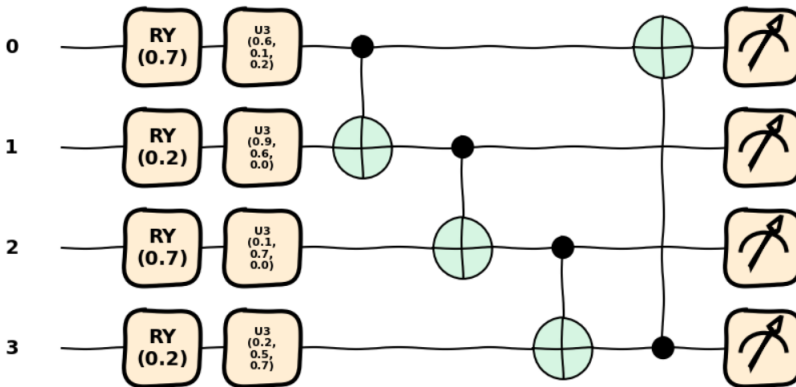


Figure 4.1: Example circuit for Observables, draw from PennyLane.

Consider a quantum circuit using  $e_2$  as the encoding layer and one  $h_6$ , see the details in Section 5.3.4, as the hidden layer, which contains 12 parameters. This circuit configuration is illustrated in Figure 4.1. When conducting experiments, we maintained consistent four inputs and initial parameters which were chosen randomly while computing the gradient of each parameter using the parameter shift rule under several different observables. The results of this experiment are detailed in Table 4.3.

The data presented in Table 4.3 highlight a crucial aspect: the choice of observable can lead to the gradient of a parameter converging to zero prematurely, even before training

Table 4.3: The influence of observable on the parameter gradient.

Observable	Gradient	Number of 0
$\sigma_x \otimes I \otimes \sigma_y \otimes I$	-0.001, -0.045, -0.028, -0.004, -0.012, -0.003, -0.008, -0.011, -0.010, -0.033, 0.012, 0.009	0
$\sigma_z \otimes \sigma_z \otimes \sigma_z \otimes \sigma_y$	0.002, 0.099, 0.062, 0.009, -0.013, -0.003, -0.019, 0.000, 0.000, 0.071, -0.027, -0.019	2
$\sigma_z \otimes \sigma_z \otimes \sigma_y \otimes \sigma_z$	-0.089, 0.000, 0.005, 0.000, 0.000, 0.000, 0.024, 0.032, 0.029, 0.099, -0.037, -0.027	4
$\sigma_y \otimes \sigma_z \otimes \sigma_y \otimes \sigma_y$	0.430, 0.000, -0.026, 0.000, 0.000, 0.000, -0.118, -0.159, -0.143, 0.000, 0.000, 0.000	7
$\sigma_z \otimes \sigma_z \otimes \sigma_z \otimes \sigma_z$	-0.656, 0.000, 0.040, 0.000, 0.000, 0.000, -0.202, 0.000, 0.001, 0.000, 0.000, 0.000,	8

has occurred. Such a scenario is generally undesirable as it may hinder the model's ability to learn and adapt effectively during the training phase.

Therefore, careful consideration and selection of observations are important in the design of quantum circuits for PVQNN. This choice not only influences the initial gradient values but also plays a vital role in the overall learning trajectory and performance of the model.

#### OPTIMIZATION HYPER-PARAMETER

The optimization procedure in PVQNN is the same as that of classical neural networks. In formulating our approach, we draw inspiration from the strategies outlined in the work of Liu et al. [7]. The key components of our optimization strategy are as follows:

##### 1. Loss function

The loss function for PVQNN is fundamentally a measure of the distance between the network's predictions and the actual data, which is quantified by the norm of their difference. In this context, we have selected the Euclidean norm, commonly referred to as the  $L_2$  norm, which is equivalent to the mean squared error (MSE). The  $L_2$  norm is highly regarded and extensively utilized as a loss function in regression problems. Its popularity stems from its proven efficacy in mitigating overfitting, thereby contributing to better generalization of the model across various datasets. As such, the loss function we utilize is expressed as follows,

$$L_2 = \text{MSE} = \frac{1}{n} \sum (y_i^{\text{out}} - y_i)^2. \quad (4.3)$$

Here,  $y_i^{\text{out}}$  represents the predicted output of the network for the  $i^{\text{th}}$  data point, and  $y_i$  is the corresponding actual value. The summation runs over all  $n$  data points in

the dataset, providing a comprehensive measure of the model's performance across the entire data spectrum.

Moreover, the choice of the  $L_2$  norm is not arbitrary but is informed by its mathematical properties. The squaring of errors in the MSE formulation inherently gives more weight to larger errors. This characteristic ensures that the model pays particular attention to data points where the prediction deviates significantly from the actual value, driving the model to achieve a more accurate fit overall.

Additionally, the  $L_2$  norm's differentiability is a critical aspect that facilitates efficient gradient-based optimization methods, which gives a great benefit to the optimization step of the model.

## 2. Optimizer

The training process for the PVQNN can be conceptualized as an optimization problem, with the primary goal being to minimize the  $L_2$  loss by fine-tuning the model parameters, denoted as  $\theta$ . This objective can be mathematically represented as,

$$\arg \min_{\theta} L_2(\theta | (\mathbf{x}, \mathbf{y})). \quad (4.4)$$

Following the methodology presented in Liu et al.'s study [7], we employ the Adaptive Moment Estimation (Adam) optimizer for this optimization task. Adam is particularly effective due to its dynamic adjustment of learning rates and its incorporation of moment-based updates, which are associated with the momentum and velocity of a particle. The update formula for Adam is given by,

$$\theta^{(t+1)} = \theta^{(t)} - \eta^{(t+1)} \frac{m^{(t+1)}}{\sqrt{v^{(t+1)} + \epsilon}},$$

where  $\eta$  represents the learning rate, and  $m$  and  $v$  are the first and second moment estimates, respectively. The update rules for these moments and the learning rate are as follows,

$$\begin{aligned} m^{(t+1)} &= \beta_1 m^{(t)} + (1 - \beta_1) \nabla L(\theta^{(t)}), \\ v^{(t+1)} &= \beta_2 v^{(t)} + (1 - \beta_2) (\nabla^2 L(\theta^{(t)})), \\ \eta^{(t+1)} &= \eta^t \frac{\sqrt{(1 - \beta_2^{t+1})}}{(1 - \beta_1^{t+1})}. \end{aligned}$$

For our implementation, the hyperparameters  $\beta_1$ ,  $\beta_2$ , and  $\epsilon$  are set to their default values of 0.9, 0.99, and  $1 \times e^{-8}$ , respectively. The gradient of the loss function, crucial for the optimization process, is computed as,

$$\nabla L = \nabla L_2(\theta) = \frac{\partial L_2(y^{out}, y)}{\partial y^{out}} \frac{\partial y^{out}}{\partial \theta}. \quad (4.5)$$

Table 4.4: The training setting of PVQNN

Setting	Option
Loss function	$L_2$ loss
Optimizer	Adam
learning rate $\eta$	0.01
$\beta_1$	0.9
$\beta_2$	0.99
$\epsilon$	$1 \times e^{-8}$

The computation of  $\frac{\partial y^{out}}{\partial \theta}$  is achieved through the parameter shift rule. We have selected a learning rate of 0.01 for the Adam optimizer, in line with the recommendations from Liu et al. [7], as this rate has been identified as optimal for problems similar to ours.

Further details about the training dataset size and batch size will be discussed in Chapter 5. It is also noteworthy to mention that, in the study by Hubregtsen et al. [43], it was found that the Adam optimizer paired with an  $L_2$  loss function yields the best results for PVQNN. This aligns with our approach and further reinforces the suitability of our chosen optimization strategy.



# 5

## RESULTS AND DISCUSSION

*In this chapter, we begin by outlining the evaluation metrics applied in our experiments and describing the dataset utilized for our research. Following this, we present the experimental results in a subsequent section. Specifically, Section 5.1 details the construction of our dataset. In Section 5.2, we discuss the results obtained from the DQNN model. Section 5.3 explores how various settings influence the performance of the PVQNN and the outcomes of the optimally tuned model.*

In this chapter, we present the performance evaluation of the DQNN and PVQNN in addressing the inverse problem of Black-Scholes models. The primary metric utilized for this assessment is the Mean Squared Error (MSE), defined as,

$$\text{MSE} = \frac{1}{n} \sum (y_i - y_i^{\text{out}})^2.$$

Here,  $y_i$  represents the actual value, and  $y_i^{\text{out}}$  denotes the predicted value generated by the model. To provide a comprehensive evaluation, additional widely recognized metrics are also employed to further gauge the models' performance. These include the Root Mean Squared Error (RMSE), the Mean Absolute Error (MAE), the Mean Absolute Percentage Error (MAPE), and the Coefficient of Determination ( $R^2$ ),

1. Root Mean Squared Error:

This is the square root of MSE, offering a measure of the standard deviation of the prediction errors. RMSE is sensitive to large errors, making it a crucial metric for understanding the performance of our models in scenarios where accuracy is paramount.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

2. Mean Absolute Error:

Representing the average of the absolute differences between the predicted and actual values, MAE provides an intuitive measure of prediction accuracy. Unlike RMSE, it does not penalize large errors as severely, thus offering a different perspective on model performance.

$$\text{MAE} = \frac{1}{n} \sum |y_i - y_i^{\text{out}}|$$

3. Mean Absolute Percentage Error:

MAPE relates the error to the actual values, offering a relative measure of accuracy. This metric is particularly useful in providing a scale-independent assessment of error magnitude.

$$\text{MAPE} = \frac{1}{n} \sum \frac{|y_i - y_i^{\text{out}}|}{y_i}$$

4. Coefficient of Determination:

$R^2$  reflects the proportion of the variance in the dependent variable explained by the independent variables in a regression model. It ranges from 0 to 1, where values closer to 1 (normally larger than 0.99) indicate a better fit of the model to the data.

$$R^2 = 1 - \frac{1}{n} \sum \frac{|y_i - y_i^{\text{out}}|^2}{|y_i - \bar{y}|^2}$$

where  $\bar{y}$  is the mean value of  $y_i$ .

These metrics collectively form a comprehensive framework for evaluating the accuracy and reliability of the DQNN and PVQNN models in their application to computing implied volatility.

## 5.1 DATASETS

For the Black-Scholes formula, a common technique in this process is the use of moneyness  $m$ , a term describing the intrinsic value of an option in its current state, defined as  $m = \frac{S}{K}$ . This metric relates the strike price to the price of the underlying asset and allows for a reformulation of the Black-Scholes formula as follows,

$$\frac{V(t,S)}{K} = \frac{S}{K} N(d_1) - e^{-r\tau} N(d_2), \quad (5.1)$$

$$d_1 = \frac{\log(\frac{S}{K}) + (r + 0.5\sigma^2)\tau}{\sigma\sqrt{\tau}}, d_2 = d_1 - \sigma\sqrt{\tau}.$$

Consequently, the inverse Black-Scholes formula can be redefined as  $\sigma = BS^{-1}(\frac{V}{K}, \tau, \frac{S}{K}, r)$ . This modification reduces the number of input features required for the neural network model, potentially simplifying the model's architecture and enhancing its efficiency.

In our research, we specifically focus on European call options. To construct a robust dataset for this instrument, we adopted the sampling methodology detailed in the study by Liu et al. [7]. This approach involves the strategic sampling of key variables: moneyness ( $S/K$ ), the risk-free interest rate ( $r$ ), the time to maturity ( $\tau$ ), and the volatility ( $\sigma$ ). These variables are sampled from their joint multidimensional distribution to reflect realistic market scenarios accurately.

For effective sampling, we employed the Latin Hypercube Sampling (LHS) method. Renowned for its efficiency in multidimensional spaces, LHS ensures a more representative and evenly distributed sample from the underlying distributions compared to traditional random sampling methods. This is achieved by dividing the range of each variable into intervals of equal probability and sampling once from each interval. Such a sampling strategy is crucial in reducing biases and over-aggregation of data, which often occur with simpler random sampling methods. This enhanced representation of the data distribution in the sampling space is pivotal in ensuring the generalizability of our models, thereby minimizing potential biases that could arise from unrepresentative or clustered samples.

Following the sampling process, we utilized the Black-Scholes formula, as indicated in Equation (5.1), to compute the scaled call price  $V/K$ . As a result, each data sample in our dataset encompasses a set of five variables:  $S/K$ ,  $r$ ,  $\tau$ ,  $\sigma$ ,  $V/K$ . Among them, we select  $S/K$ ,  $r$ ,  $\tau$ ,  $V/K$  as the feature set  $x$  for the input to the DQNN and PVQNN. The volatility  $\sigma$  is designated as the label  $y$ . The detailed composition and structure of the dataset are further elucidated in Table 5.1, providing a clear overview of the data used in our analysis.

Table 5.1: Data set

QNN	Parameters	Range	Unit
Input	moneyness $S/K$	[0.98, 1.02]	-
	Time to Maturity $\tau$	[0.5, 0.6]	year
	Risk free rate $r$	[0.03, 0.08]	-
	Scaled call price $V/K$	(0.10, 0.22)	-
Output	Volatility $\sigma$	[0.3, 0.7]	-

Another dataset, which incorporates gradient-squashing adjustment, is detailed in Table 5.2.

Table 5.2: Data set after scaling.

QNN	Parameters	Range	Unit
Input	moneyness $S/K$	[0.98, 1.02]	-
	Time to Maturity $\tau$	[0.5, 0.6]	year
	Risk free rate $r$	[0.03, 0.08]	-
	Scaled time value $\log(\tilde{V}/K)$	(-2.7,-1.7)	-
Output	Volatility $\sigma$	[0.3, 0.7]	-

## 5.2 DQNN

Building upon the experimental settings detailed in Section 4.2.2 and the dataset setup in Table 5.2, we examine the model performance in computing the implied volatility based on the model in [44]. The graph in Figure 5.1 illustrates the trajectory of the training loss against the number of training steps.

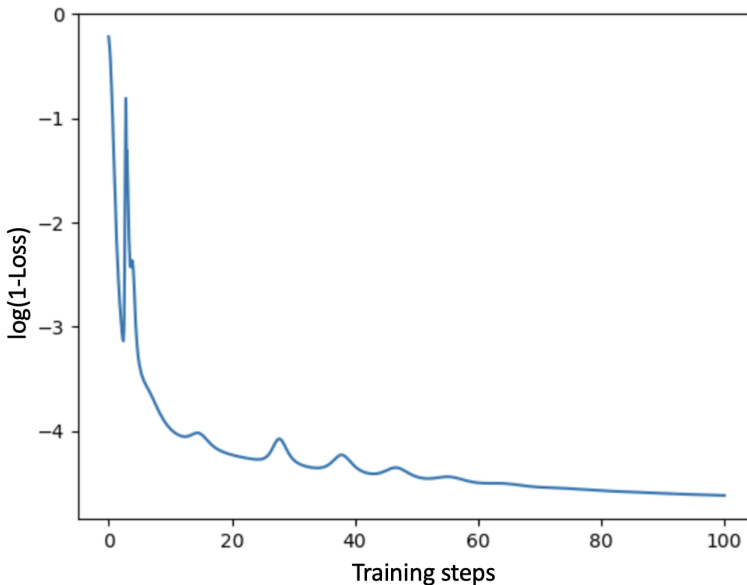


Figure 5.1: Training loss over training steps for computing the implied volatility

The training loss here is the transformation of one minus fidelity value after a logarithmic adjustment  $\log(1-L)$ . For visual clarity, the training steps are scaled down by a factor of 10. This rescaling makes the graphical representation more comprehensible without

altering the underlying trend. Furthermore, the model's performance can be quantitatively assessed through the metrics presented in Table 5.3.

Table 5.3: The performance for the trained DQNN.

	Fidelity	MSE	RMSE	MAE	MAPE	$R^2$
Train (160)	0.99016	0.00503	0.07093	0.05569	0.07452	0.47026
Test (40)	-	0.00511	0.07093	0.05551	0.07532	0.46261

As depicted in the table, the model achieves commendable performance on the fidelity loss function, exceeding 0.99 on the training set. However, there appears to be a discrepancy when it comes to the classical metrics of MSE, RMSE, MAE, MAPE, and  $R^2$ . The decoding process of the output quantum state  $\rho_{out}$  to classical data  $y_{out}$ , necessary to compute these metrics, has introduced discrepancies.

Since in our practice, the label data  $\sigma$ , representing volatility, is encoded into a quantum state  $\rho_{label}$  using a rotation gate Ry. This operation translates the classical information into the quantum state as follows,

$$\rho_{label} = |\phi^{label}\rangle\langle\phi^{label}| = \begin{bmatrix} \cos(\sigma) \\ \sin(\sigma) \end{bmatrix} \begin{bmatrix} \cos(\sigma) & \sin(\sigma) \end{bmatrix} = \begin{bmatrix} \cos^2(\sigma) & \cos(\sigma)\sin(\sigma) \\ \cos(\sigma)\sin(\sigma) & \sin^2(\sigma) \end{bmatrix}.$$

The resulting quantum state  $\rho_{label}$  is a pure state represented in the density matrix form, which corresponds to the output quantum state from the DQNN  $\rho_{out}$ . The process of decoding  $\rho_{out}$  involves extracting the classical information  $\sigma^{out}$ . This is achieved by taking the real part of the first entry of  $\rho_{out}$ ,  $\cos^2\sigma$ , then computing its square root to obtain  $\cos\sigma$ , and finally applying the arccos function to retrieve the estimated volatility  $\sigma^{out}$ .

This decoding process, however, comes with these limitations:

1. **Imaginary Component Disregard:** By taking only the real part of the first entry of  $\rho_{out}$ , the process neglects any imaginary components, which could contain information in the quantum state.
2. **Information Loss:** The decoding uses only one element of the density matrix, thus discarding information from the other elements.
3. **Precision Loss:** The extraction of  $\rho_{out}$  from  $\cos^2(\sigma)$  is subject to computational precision limits. The arccos function, particularly near its bounds, can be sensitive to small changes in its argument, potentially amplifying any prior computational errors.

This phenomenon is also evident in the work of Sakuma [19]. In his study, he designates the strike price  $K$  as the input variable, while keeping the other four parameters as constants. The model's output is the implied volatility  $\sigma$ . Notably, one of his models, which is a 1-2-1 DQNN attained a high fidelity loss score of 0.998 after training. However, it exhibited an MSE of approximately 0.003 on the testing data set. This outcome aligns with the limitations previously discussed in our analysis.

These factors highlight the complicated balance in quantum-classical interfaces, where information-rich quantum representations are translated back into classical outputs. The precision and accuracy of this decoding are important, as they directly affect the model's performance metrics. Future improvements may involve exploring alternative decoding schemes that can capture more information from the quantum state or employing error mitigation techniques to enhance the fidelity of the decoding process.

Furthermore, the training time of this DQNN model on a classical CPU simulator was 18,359 seconds in total on the CPU (Intel i7-10750H, 2.60GHz), which suggests that quantum simulation requires more computational intensity compared to classical neural networks.

Another important consideration in our study is the optimization process of DQNN. This process necessitates the retrieval of the quantum state after each layer to update parameters accordingly. However, with current NISQ devices, this step presents a non-trivial challenge. The act of measurement in quantum systems inherently alters the state of the qubits, which is a phenomenon known as quantum collapse. This characteristic of quantum measurement prevents the direct observation of intermediate states without disrupting the computation, making the DQNN optimization process incompatible with the current device. Given these limitations, our implementations are more toward the PVQNN.

5

### 5.3 PVQNN

In this section, we assess the PVQNN model, initially examining the effect of training data size on performance. This is followed by an exploration of key structural elements that impact its efficiency, including data encoding strategies, the architecture of hidden layers, and the application of data re-uploading techniques. Our objective is to refine the model for effective computation of implied volatility.

#### 5.3.1 DATA SIZE

To explore the relationship between prediction accuracy and training set size, our study examined a range of training data set sizes: 4, 8, 12, 16, 20, 40, 100, 200, 400, 800, 1600, 4000, 8000, and 16000 samples. These sets were used to train a specifically designed PVQNN model. This model featured an  $e_2$  data encoding layer, see Table 5.7, three hidden layers utilizing the  $h_6$  strategy, see Table 5.13, and the implementation of data-reuploading techniques. In the measurement layer,  $\sigma_z \otimes \sigma_z \otimes \sigma_z \otimes \sigma_z$  observables were employed.

Table 5.4: Data set for data size experiments.

QNN	Parameters	Range	Unit
Input	moneyness $S/K$	[0.95, 1.05]	-
	Time to Maturity $\tau$	[0.5, 0.6]	year
	Risk free rate $r$	[0.02, 0.05]	-
	Scaled time value $\log(\tilde{V}/K)$	(-2.8,-1.6)	-
Output	Volatility $\sigma$	[0.3, 0.7]	-

Regarding the data parameters, slight adjustments were made as outlined in Table 5.4. For training data sizes larger than or equal to 100 samples, a batch size of 64 was used,

while a smaller batch size of 4 was chosen for fewer samples. When dealing with larger data sets (sizes greater than 100), training was halted after 150 epochs, a point at which sufficient model convergence was typically achieved. Conversely, for training sets less than 100 samples, a different training duration was adopted. Post-training, the effectiveness of each model was assessed using the same testing data set comprising 4000 samples. These samples were selected from the same distribution as that of the training data sets.

Table 5.5: Experimental settings for data size

Parameter	Value
Training Data Sizes	4, 8, 12, 16, 20, 40, 100, 200, 400, 800, 1600, 4000, 8000, 16000
Testing Data Size	4000
PVQNN Configuration	
Encoding Layer	$e_2$
Ansatz Strategy	$h_6$
Hidden Layers	3
Measurement	$\sigma_z^{\otimes 4}$
Training Details	
Epochs	More than 1000 for training sizes under 100 150 for other
Batch Size	64 for training sizes above 100 4 for other

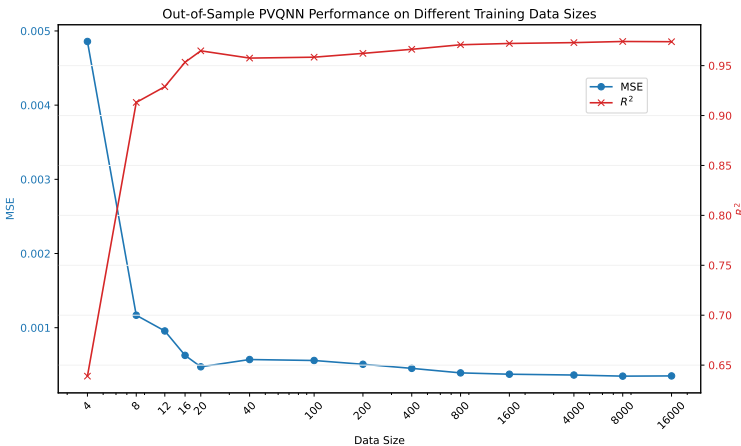


Figure 5.2: Out-of-Sample PVQNN performance on different training data sizes

The comprehensive experimental setup, including all parameters and configurations used, can be found in Table 5.5. The outcomes of these experiments, highlighting the

relationship between training data size and model accuracy, are depicted in Figure 5.2, and more details can be found in Appendix A. This structured approach allowed for a nuanced understanding of how varying training data volumes impact the predictive capabilities of the PVQNN model.

The analysis of different training data sizes reveals interesting insights into the model's performance. With just 16 training data, the model demonstrates promising results. However, as we reduce the training data size to less than 40, we observe clear signs of different levels of overfitting during the training process. This phenomenon is particularly pronounced with a training data size of 5. As we increase the amount of training data, the model's performance on the training set improves, but simultaneously, its performance on the testing set degenerates.

To mitigate the potential influence of initial parameter settings, we conducted a systematic exploration of random seed values. The default random seed, set at 95, was retained as a baseline. Four additional seed values of 12, 27, 66, and 88 were selected for experimentation. The primary objective was to investigate the impact of these different random seed values on model performance. Each training run consisted of 20 data. The training procedure was conducted over 1000 epochs, ensuring the model converges. See the result in Table 5.6.

Table 5.6: Out-of-Sample PVQNN performance for different initialization seeds.

Random Seed	MSE	$R^2$
95	0.000627	0.953397
12	0.000586	0.956449
27	0.000486	0.963865
66	0.000583	0.956662
88	0.000540	0.959846

Upon analyzing the results, it is evident that all configurations yielded satisfactory outcomes, all the  $R^2$  are above 0.95. Furthermore, it is worth noting that by fine-tuning the initial parameters, some enhancements in the model's performance can be achieved, for example, by choosing the random seed as 27, the MSE decreases by about 22.5% and  $R^2$  increases by 0.01 compared with the baseline setting.

In the study by Matthias et al. [45], a similar observation was made concerning the training data size required for achieving desirable performance in PVQNNs. Their work also suggests that the amount of training data only needs to be similar to the total number of parameters in the PVQNN model to achieve an efficient generalization performance. In our research, we employ an  $h_6$  ansatz structure, repeated three times, resulting in 36 parameters. Consequently, our findings align with their work, indicating that a training dataset exceeding 36 mitigates the risk of overfitting, a hypothesis corroborated by our results where datasets smaller than 40 exhibited overfitting issues.

The potential for QNNs to successfully train complex models with substantially smaller datasets compared to classical neural networks may be attributed to the unique data encoding methods. In classical computing, a single feature of input data is typically represented in a one-dimensional format. However, in QNNs, This feature undergoes data coding, a process that maps it into a qubit, whose Hilbert space is the  $C^2$ .



In this Hilbert space, operations occur in a multidimensional context, which contrasts with the linear, one-dimensional operations in classical neural networks. This increased dimensionality in feature space could theoretically enable more complex patterns to be identified and learned from a smaller dataset, as the quantum system can encapsulate and process information more densely and efficiently, like the kernel methods that have been widely applied in classical machine learning [36].

Future research should focus on validating this explanation that the efficacy of training PVQNN networks with reduced data correlates with the increasing of feature space dimensions through quantum coding. Additionally, exploring the limits and capabilities of quantum data encoding, and comparing its efficacy directly with traditional methods in various scenarios is also needed.

In summary, for PVQNN, we are able to train our model with a much smaller amount of training data compared to that used for traditional neural networks without experiencing overfitting. This observation has important implications for our ability to train models at a fast pace and in situations where there is a lack of training data.

### 5.3.2 DATA ENCODING

To examine how different encoding methods affect the prediction accuracy of PVQNN, we conducted a series of experiments using various encoding techniques.

#### ENCODING STRATEGIES

In our research, we experimented with nine different types,  $e_1, \dots, e_9$ , of Time-evolution encoding methods to ascertain the most suitable one for our specific application. The input data  $\bar{x}$  is encoded based on these nine strategies. Here we use four input features  $\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4$  to construct these strategies. Among them, all strategies encode data to four qubits except  $e_4$  which uses only two qubits for data encoding, which has been applied in DQNN. This alignment of qubits with input features simplifies the encoding process and allows for the exploration of various ansatz structures. Additionally, it ensures that the size of the model remains within the limits that current quantum computing devices can support.

Based on the characteristics of these strategies we classify them into three categories, as detailed in Table 5.7.

These methods were chosen based on their potential compatibility with the nature of our data and the constraints of current quantum computing technology. The detailed analysis and comparison of these encoding techniques are provided in the subsequent part. This comprehensive exploration allows us to identify the most effective encoding strategy for optimizing the performance of quantum neural network algorithms in handling decimal, small-size feature datasets.

#### 1. Simple Angle Encoding

Here we employed three primary quantum rotation gates: Rx, Ry, and Rz. These gates are associated with rotations around the x, y, and z axes, respectively, and can be expressed as the evolution time of some specific Hamiltonians  $\sigma_x$ ,  $\sigma_y$ , and  $\sigma_z$ , see Table 3.1. For detailed implementations, the input data  $\bar{x}(\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4)$  will be directly used as input angles passing to the rotation gates, see Table 5.7 part1 which outlines the correspondence between each gate and the input.

For example, for  $e_2$ , the state of encoded qubits can be expressed in this form,

$$\begin{aligned}
 |\bar{x}\rangle &= \bigotimes_{i=1}^4 R_y(\bar{x}_i)|0\rangle \\
 &= \bigotimes_{i=1}^4 \begin{bmatrix} \cos(\bar{x}_i/2) & -\sin(\bar{x}_i/2) \\ \sin(\bar{x}_i/2) & \cos(\bar{x}_i/2) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\
 &= \bigotimes_{i=1}^4 \cos(\bar{x}_i/2)|0\rangle + \sin(\bar{x}_i/2)|1\rangle.
 \end{aligned}$$

Table 5.7: Encoding strategies (draw from PennyLane)

5

		Part 1:			
Strategy Number		$e_1$	$e_2$	$e_3$	$e_4$
Circuit	0				
	1				
	2				
	3				
		Part 2:			
Strategy Number		$e_5$	$e_6$	$e_7$	
Circuit	0				
	1				
	2				
	3				
		Part 3:			
Strategy Number		$e_8$	$e_9$		
Circuit	0				
	1				
	2				
	3				

## 2. Dense Angle Encoding

Dense angle encoding represents an advanced development of simple angle encoding. Its primary advantage lies in its efficiency in qubit usage. Specifically, it reduces the number of required qubits to half ( $N/2$ ) for encoding an equivalent number of input features compared to other methods. This reduction is particularly beneficial in the NISQ applications, as it allows for more compact neural network architectures and potentially reduces quantum error rates, as noted by LaRose [46].

The mathematical formulation of dense angle encoding is given by,

$$|\bar{x}\rangle = \bigotimes_{k=1}^{\lfloor N/2 \rfloor} \cos(\pi \bar{x}_{2k-1})|0\rangle + e^{2\pi i \bar{x}_k} \sin(\pi \bar{x}_{2k-1})|1\rangle.$$

In terms of circuit implementation, this encoding strategy is facilitated by the U3 rotation gate. An illustrative example of this implementation can be seen in the encoding strategy  $e_4$ .

## 3. Simple Angle Encoding Variation

Here, we explored three distinct encoding strategies derived from the simple angle encoding method. The first strategy, labeled  $e_5$ , involves encoding the input data  $2\pi\bar{x}$  directly into the Rx rotation gate. This method interests the relationship between the size of input data and the model output, facilitating a straightforward encoding process.

The second strategy,  $e_6$ , is a more complex encoding approach. Initially, it encodes the input data  $\bar{x}$  into the Rx rotation gate. Subsequently, the qubit undergoes additional rotations around the y and z axes, each by an angle of  $\frac{\pi}{4}$ . This particular strategy was employed in the study conducted by Hubregtsen et al. [43], which ensures the encoded data maintains a consistent range across different computational bases. Such uniformity is pivotal for achieving an unbiased embedding when using the simple angle encoding method.

The third strategy,  $e_7$ , combines the Hadamard gate with the Rx rotation gate, using  $\bar{x}$  as the input. This combination introduces a superposition state before the rotation, potentially enhancing the ability to encode representations.

## 4. ZZFeature Map

The ZZFeature Map, also known as Instantaneous Quantum Polynomial (IQP) Encoding, was proposed by Havlíček et al. [47]. This method has demonstrated significant efficacy in classification problems within quantum computing. The process initiates with a series of Hadamard gates applied to all qubits, creating a superposition. Subsequently, Rz rotation gates are applied, with the input data  $\bar{x}$  as the rotation angles. The distinctive feature of this method is the sequential entanglement of all qubits, employing controlled-NOT gates and Rz rotation gates. The mathematical expression of this encoding is,

$$|\bar{x}\rangle = (U_Z(\phi(\bar{x}))H^{\otimes n})^r |0^n\rangle.$$

Here,  $U_Z(\bar{\mathbf{x}})$  represents the unitary operation combining pairwise Z rotations and individual Z rotations,

$$U_Z(\bar{\mathbf{x}}) = \prod_{[i,j] \in S} R_{Z_i Z_j}(\phi(\bar{x}_i, \bar{x}_j)) \bigotimes_{k=1}^n R_z(\phi(\bar{x}_k)),$$

where the non-linear function  $\phi(\bar{x})$  is defined as,

$$\phi(\bar{x}) = \begin{cases} \bar{x}_i & \bar{x} = \bar{x}_i \\ (\pi - \bar{x}_i)(\pi - \bar{x}_j) & \bar{x} = \bar{x}_i, \bar{x}_j \end{cases}$$

In this formulation,  $r$  denotes the depth of the encoding circuit, indicating the number of repetitions, and  $n$  represents the number of qubits involved. The set  $S$  defines the entanglement pattern; for instance, in encoding strategy  $e_8$ , the entangling gates connect neighboring qubit pairs.

Furthermore, the work of Du et al. [40] introduces a variation of the ZZFeature Map. This variant, illustrated in encoding strategy  $e_9$ , starts with Hadamard gates followed by Ry rotation gates, using  $\bar{x}$  as the input. The entanglement process then involves sequentially entangling qubits with controlled rotation y gates, each gate's rotation angle determined by  $(\pi - \bar{x}_1)(\pi - \bar{x}_2)(\pi - \bar{x}_3)(\pi - \bar{x}_4)$ . This adaptation demonstrates the flexibility and potential for customization inherent in the ZZFeature Map approach.

## EXPERIMENTS

Specifically  $e_1, e_2, e_3, e_5, e_6, e_7, e_8$ , and  $e_9$  are used for PVQNN. Each PVQNN model was configured with the same hidden layer ( $h_6$ , see in Section 5.3.4) and used identical observables ( $\sigma_x \otimes I \otimes \sigma_y \otimes I$ ) in the measurement layer. This uniformity across models allowed for a controlled comparison of the encoding methods' effectiveness.

Our experimental procedure involved training each model with a dataset of 80 data points, followed by testing on a distinct set of 20 data points, since this amount number training data is enough to train a satisfied model from the previous experiments. This process was replicated across 300 epochs, maintaining a consistent batch size of 20 for each epoch.

Furthermore, acknowledging the unique aspect of  $e_5$  encoding as a scaling mechanism for input data, we introduced an additional encoding strategy,  $e_{10}$ . This strategy omits the gradient-squash approach during the input data processing, as outlined in Table 5.1. Subsequently,  $e_{10}$  encoding was employed to convert the data into quantum states. This variation aims to assess the impact of the gradient-squash approach on the model's performance.

Details of the experimental setup, including the PVQNN model configurations and the characteristics of each encoding method, are comprehensively documented in Table 5.8.

Table 5.8: Experimental settings for data encoding methods.

Parameter	Value
Dataset	Training: 80, Testing: 20
PVQNN Configuration	
Encoding Layer	$e_1, e_2, e_3, e_5, e_6, e_7, e_8, e_9, e_{10}$
Hidden Layer	$h_6$
Observables	$\sigma_x \otimes I \otimes \sigma_y \otimes I$
Training Details	
Epochs	300
Batch Size	20

The main outcomes of these experiments, demonstrating how encoding strategy impacts the accuracy of PVQNN models, are presented in Table 5.9, and full results can be found in Appendix A.

5

Table 5.9: Out-of-Sample PVQNN performance with different data encoding methods.

Data Encoding	MSE	$R^2$
$e_1$	0.001139	0.91578
$e_2$	0.001139	0.91577
$e_5$	0.223	-15.44
$e_{10}$	0.00802	0.409
$e_6$	0.001138	0.91612
$e_8$	0.0007478	0.94490

Analyzing the data from Table 5.9, it is evident that encoding methods  $e_1$ ,  $e_2$ ,  $e_6$ , and  $e_8$  demonstrate exceptional performance characterized by low MSE and high  $R^2$  values. This pattern suggests that these methods are particularly adept at minimizing predictive errors and effectively capturing the variance in our PVQNN model.

Noteworthy, a comparison of  $e_5$  and  $e_{10}$  with  $e_1$  — all of which utilize the Rx gate for encoding — reveals a significant impact of input data scaling on the model's output. Notably, encoding  $e_5$ , which scales the input data by a factor of  $2\pi$  relative to  $e_1$ , exhibits markedly inferior performance in the model. Moreover, incorporating gradient-squash techniques enhances predictive accuracy, aligning with findings by Liu et al. [7]. This highlights the fact that the size of the input values has a significant impact on the model. Therefore, we need to choose the appropriate data scaling methods to preprocess the input data. Additionally, this phenomenon could also be attributed to the nature of quantum encoding in projecting data into a higher dimensional feature space, thereby influencing the relative distances among differently sized data points.

In conclusion, the findings from Table 5.9 not only underscore the significance of selecting appropriate encoding methods for optimal PVQNN performance but also shed

light on the unique characteristics of these quantum neural networks.

### 5.3.3 DATA RE-UPLOADING AND NUMBER OF HIDDEN LAYERS

To explore how Data Re-uploading and the variation in the number of hidden layers affect the prediction accuracy of PVQNN, we designed an experiment with a specific focus on these two parameters. Our approach involved employing the  $e_2$  encoding method across different PVQNN models. Each model varied only in the number of hidden layers and whether to adopt data re-uploading while maintaining a consistent layer construction strategy ( $h_6$ , see in Section 5.3.4) and the same observables ( $\sigma_x \otimes I \otimes \sigma_y \otimes I$ ) in the measurement layer.

The experimental setup, which is clearly outlined in Table 5.10, included training each PVQNN model with a dataset comprising 40 data points, followed by testing the model's performance on a distinct set of 10 data points. This process was iteratively conducted for 500 epochs, with a batch size of 20. The aim was to ensure comprehensive and reliable insights into the models' performance under different hidden layer configurations.

Table 5.10: Experimental settings for PVQNN with varying hidden layers.

Setting	Value
Dataset	Training: 40, Testing: 10
<b>PVQNN Configuration</b>	
Encoding Layer	$e_2$
Hidden Layer Type	$h_6$
Number of Hidden Layers	1, 2, 3, 4, 5
Observables	$\sigma_x \otimes I \otimes \sigma_y \otimes I$
<b>Training Details</b>	
Epochs	500
Batch Size	20

Results from this experiment, providing insights into the impact of hidden layer variations and Data Re-uploading on the prediction accuracy of PVQNN models, are compiled in Table 5.11.

Notably, there is a general improvement in the model's performance with an increasing number of hidden layers. However, it is observed that beyond two layers, the extent of this enhancement becomes less pronounced. This suggests a potential point of diminishing returns in adding more hidden layers, indicating that beyond a certain level, additional complexity does not substantially contribute to performance improvements.

Furthermore, the inclusion of data re-uploading stands out as a significant factor in enhancing the model's performance. For each given number of hidden layers, the incorporation of data re-uploading consistently results in lower MSE and higher  $R^2$  values compared to models without data re-uploading. This improvement in  $R^2$  is quantified at an average of approximately 2.77%. This figure highlights the effectiveness of data re-uploading in refining the model's predictive accuracy and its capability to account for variance within the data.

Table 5.11: Out-of-Sample PVQNN performance based on Hidden Layer number and Data Re-uploading

Hidden Layer number	Data Re-uploading	MSE	$R^2$
1	No	0.001404	0.90249
2	No	$4.978 \times 10^{-4}$	0.96544
2	Yes	$1.852 \times 10^{-4}$	0.98714
3	No	$4.288 \times 10^{-4}$	0.97023
3	Yes	$1.667 \times 10^{-5}$	0.99884
4	No	$4.439 \times 10^{-4}$	0.96918
4	Yes	$8.226 \times 10^{-6}$	0.99943
5	No	$3.366 \times 10^{-4}$	0.97662

These findings are in concordance with the conclusions drawn by Schuld et al. [39]. Specifically, they indicate that merely increasing the number of hidden layers has a negligible impact on the distribution of Fourier coefficients delineated in Equation (3.20). Consequently, this augmentation does not significantly influence the model's performance. In contrast, employing data re-uploading techniques can linearly expand the frequency spectrum  $\Omega$ . Such expansion enhances the expressivity of the PVQNN model, thereby contributing to an improvement in its predictive performance.

In extending this analysis, it's important to consider the implications of these findings for practical applications of PVQNN. The optimal number of hidden layers and the decision to include data re-uploading should be tailored to the specific requirements and constraints of the application in question. More hidden layers and data re-uploading can boost performance but also increase computational complexity. In particular, additional hidden layers usually mean more training parameters, leading to longer training times in simulators. Therefore, these factors should be carefully weighed against the computational resources available and the specific needs of the application, such as the level of predictive accuracy required and the complexity of the data being modeled.

In summary, Table 5.11 provides valuable insights into the architectural optimization of PVQNN models. It highlights the importance of a balanced approach that considers both the structural complexity of the model and the advanced data processing techniques to achieve optimal performance.

### 5.3.4 ANSATZ STRUCTURE

For this part, our investigation focused on understanding the impact of ansatz structures on prediction accuracy within PVQNN. A comprehensive study was conducted using a specific encoding method and varying ansatz structure.

#### ANSATZ DESIGNS

In our study, we have proposed 15 distinct module designs. These designs vary in terms of entanglement strategies and the universality of gates. The details of these designs are as follows:

1. Entanglement Type

We have devised ansatz modules that incorporate U3 gates, capable of represent-

ing any arbitrary single-qubit gates. By altering the number and arrangement of CNOT gates, we investigate how different entanglement approaches affect model performance. Specific variations ( $h_1, h_2, h_3, h_4, h_6, h_9$ ) differ in the number of CNOT gates used. Furthermore, designs  $h_4$  and  $h_5$ ;  $h_6, h_7$ , and  $h_8$  vary in the entanglement pattern between qubits. Each module in this category has 12 parameters. The detailed configurations are presented in Tables 5.12 and 5.13.

Table 5.12: Ansatz strategies (part 1, draw from PennyLane)

strategy number	$h_1$	$h_2$	$h_3$	$h_4$	$h_5$
entanglement type	none	single	Two	chain	pyramid
circuit					

5

Table 5.13: Ansatz strategies (part 2, draw from PennyLane)

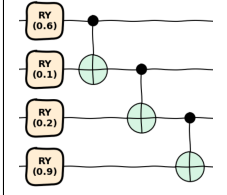
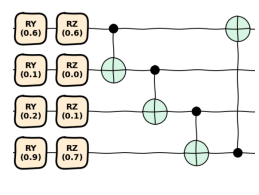
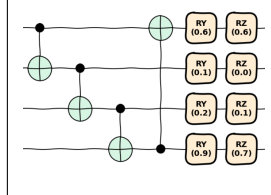
strategy number	$h_6$	$h_7$	$h_8$	$h_9$
entanglement type	ring	inverse ring	random	all-to-all
circuit				

## 2. Number of Parameters

This aspect focuses on modifying gates with parameters, such as replacing the U3 gate with Ry and Rz gates. We also explore the effects of altering the sequence between parameterized gates and entanglement gates.



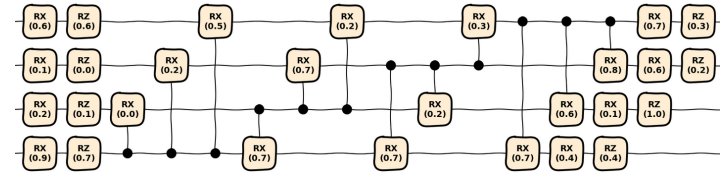
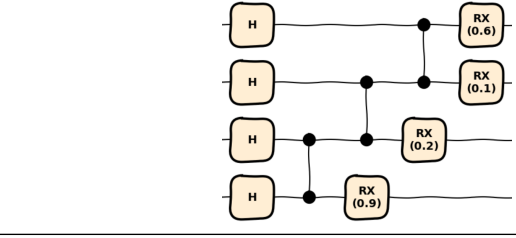
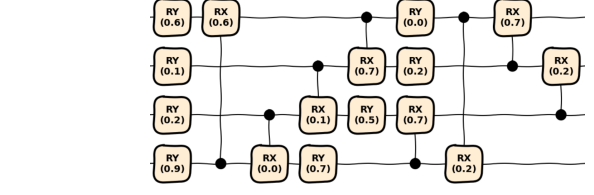
Table 5.14: Ansatz strategies (part 3, draw from PennyLane)

strategy number	$h_{10}$	$h_{11}$	$h_{12}$
parameter number	4	8	8
circuit			

3. Other considerations

We have also contained three circuit templates from the work of Sim et al. [48]. These templates (circuits 6, 9, and 14 from their works) are recognized for their potential to achieve high expressibility and improve model performance. The structures of these templates are detailed in Table 5.15.

Table 5.15: Ansatz strategies (part 4, draw from PennyLane)

strategy number	parameter number	circuit
$h_{13}$	28	
$h_{14}$	4	
$h_{15}$	20	

## EXPERIMENTS

Primarily, we employed the encoding method  $e_2$  across our experiments. However, for the  $h_{10}$  ansatz structure, we opted for  $e_1$  as the encoding strategy due to a limitation with  $e_2$  causing gradients to become zero, rendering the model untrainable.

Our experimental design involved configuring PVQNN models with a two-layer hidden layer setup and testing 15 distinct ansatz structures. Also, considering the parameters number and their performance in [48],  $h_{13}$  and  $h_{15}$  only repeated once, and  $h_{14}$  repeated five times.

Despite the variation in ansatz structures, we maintained consistency in the measurement layer by using the same observables ( $\sigma_x \otimes I \otimes \sigma_y \otimes I$ ) for all models. For each model configuration, we trained the network with 40 data and then tested its performance with 10 data. This process was replicated for 300 epochs for each model, with a batch size of 20, to ensure a robust evaluation of the model's predictive capabilities. Details of this elaborate experimental setup can be found in Table 5.16.

Table 5.16: Experimental settings for PVQNN ansatz structure

Settings	Value
Dataset Split	Training: 40, Testing: 10
<b>PVQNN Configuration</b>	
Encoding Layer	$e_2$ ( $e_1$ for $h_{10}$ )
Hidden Layer Type	$h_1, h_2, h_3, \dots, h_{15}$
Number of Hidden Layers	2 (5 for $h_{14}$ ; 1 for $h_{13}, h_{15}$ )
Observables	$\sigma_x \otimes I \otimes \sigma_y \otimes I$
<b>Training Details</b>	
Epochs	300
Batch Size	20

The main findings, as detailed in Table 5.17, provide insightful observations regarding the impact of ansatz structure on the performance of PVQNN. In the table, 'G' represents the count of parameters whose gradients are zero before the start of training. This metric offers a preliminary indication of the model's potential expressibility and trainability.

Analysis of the results reveals a significant influence of the chosen ansatz structure on the model's accuracy and fit. A majority of the models achieve an  $R^2$  value greater than 0.9, with structures  $h_7, h_9$ , and  $h_{11}$  standing out for their particularly promising outcomes. This high  $R^2$  value indicates a strong explanatory power and fit to the data.

From the perspective of entanglement, the ansatz structures  $h_1, h_2, h_3, h_4, h_7$ , and  $h_9$  differ in their number of CNOT gates, which affects their expressibility. Structures  $h_1$  and  $h_2$ , with fewer than two CNOT gates, exhibit poor expressibility, as evidenced by their relatively high MSE values (greater than 0.01). In contrast, other structures, particularly  $h_9$ , which boasts the highest entangling power, demonstrate more favorable results.

Comparing specific pairs of structures, such as  $h_4$  and  $h_5$ ;  $h_6, h_7$ , and  $h_8$ ; and  $h_{11}$  and  $h_{12}$ , it becomes apparent that the sequence of entangling specific qubits is not as critical as

Table 5.17: Out-of-Sample PVQNN performance across different ansatz structures

Ansatz Structure	Parameters	G	MSE	$R^2$
$h_1$	24	12	0.01412	0.01977
$h_2$	24	7	0.01413	0.01856
$h_3$	24	0	$7.212 \times 10^{-4}$	0.94992
$h_4$	24	0	0.00111	0.92259
$h_5$	24	0	$7.443 \times 10^{-4}$	0.94831
$h_6$	24	0	$7.428 \times 10^{-4}$	0.94841
$h_7$	24	2	$6.608 \times 10^{-4}$	0.95411
$h_8$	24	1	$8.761 \times 10^{-4}$	0.93916
$h_9$	24	4	$5.357 \times 10^{-4}$	0.96280
$h_{10} (e_1)$	24	12	0.00413	0.71299
$h_{10} (e_2)$	24	24	-	-
$h_{11}$	16	0	$6.779 \times 10^{-4}$	0.95293
$h_{12}$	16	4	$8.720 \times 10^{-4}$	0.93945

the number of CNOT gates. This insight suggests that while the depth of entanglement is important, the specific order of entanglement operations has a lesser impact on model performance.

A comparison of  $h_{10}$  and  $h_4$ ;  $h_6$  and  $h_{11}$  indicates that the choice of parameterized gates, while relevant, is not the most crucial factor. However, the number of trainable parameters (total parameters number minus G) seems to be significantly important, as seen in the results for  $h_1$ ,  $h_2$ , and  $h_{10}$ .

In summary, to achieve the desired model performance, a PVQNN's ansatz must ensure sufficient entanglement among qubits and possess an adequate number of trainable parameters. This combination is essential for enhancing the model's expressibility and ensuring its effective training and performance. Therefore, careful selection and optimization of the ansatz structure, focusing on entanglement depth and trainable parameters, are key to developing efficient and accurate PVQNN models.

### 5.3.5 OPTIMAL MODEL

In this section, we concentrate on identifying the optimal model for our research problem.

Based on the outcomes of our previous experiments and considering factors such as training time, we have selected a combination of strategies to establish the most effective model structure. Our first approach integrates the superior encoding strategy ( $e_8$ ) with the most effective hidden layer ansatz ( $h_9$ ). The second strategy employs a relatively efficient encoding strategy ( $e_6$ ) with a hidden layer ansatz ( $h_{11}$ ), which has fewer trainable parameters. For this approach, we have modified the entanglement strategy of  $h_{11}$  from a ring pattern to an all-to-all pattern, as this has shown improved performance. The third strategy is based on achieving the best results from our previous experiments, detailed in Table 5.18.

To adapt to the convergence situation and training needs, we have set varying numbers of training epochs and sizes of training data (200 for models 1 and 2, 50 for model 3) for

Table 5.18: Configuration settings of three PVQNN models

Number	Circuit	Encoding	Ansatz	Observable	Hidden Layer Count	G	Parameter Count
1	$(e_8+h_9) \times 3$	$e_8$	$h_9$	$\sigma_x \otimes I \otimes \sigma_y \otimes I$	3	4	36
2	$(e_6+h_{11}^*) \times 3$	$e_6$	$h_{11}^*$	$\sigma_x \otimes I \otimes \sigma_y \otimes I$	3	3	24
3	$(e_2+h_6) \times 4$	$e_2$	$h_6$	$\sigma_x \otimes I \otimes \sigma_y \otimes I$	4	0	48

these three models. All models are trained using the same data range, as specified in Table 5.2, and are tested on a dataset comprising 5000 data points generated from the same distribution. We employed the Adam optimizer with a learning rate  $\lambda$  of 0.1 over 500 epochs. The outcomes of these experiments are presented in Table 5.19.

Table 5.19: Out-of-Sample performance of three PVQNN models

Model Number	Training Data Size	Batch Size	MSE	RMSE	MAE	MAPE	$R^2$
1	160	64	0.0002599	0.01612	0.01145	0.023626	0.98063
2	160	64	0.0001528	0.01236	0.01011	0.020973	0.98862
3	40	20	$1.503 \times 10^{-5}$	$3.88 \times 10^{-3}$	$3.02 \times 10^{-3}$	$6.36 \times 10^{-3}$	0.99888

Analysis of the data presented in Table 5.19 reveals that all three PVQNN models achieved notable results, yet there are distinct differences in their performance levels. The third model, with its lowest values in error metrics and a higher  $R^2$  0.99888, clearly outperforms the others.

Comparatively, the second model also shows an improvement over the first, evidenced by its lower error metrics and a higher  $R^2$  value of 0.98862. This suggests that the model's efficient encoding strategy coupled with a less complex hidden layer ansatz can be more effective than merely using the best individual components, as was the case in the first model.

These findings highlight a crucial insight: the combination of individually high-performing layers does not automatically translate into the most effective overall model. The synergy between different components of a model plays a significant role in its overall performance. This is particularly evident in the case of Model 3, where a balanced approach to both structure and training parameters has yielded the most accurate and reliable results.

Therefore, it is crucial to adopt an integrative approach when selecting a model for a specific problem. A model must have high-performing individual components and demonstrate an optimized structure and parameter setting that aligns with the problem's unique requirements. Future explorations could benefit from this understanding, focusing on the fine-tuning of model structures and parameters to achieve the most effective and efficient solutions.

In addition, our model also performs well on a larger range dataset, see the Appendix A for more details.

## 5.4 SUMMARIZATION

Our numerical result research explores the performance of DQNN and PVQNN for computing implied volatility in various contexts. Key findings include:

### 1. DQNN Performance and Challenges:

The DQNN model exhibits good performance in terms of the fidelity loss function, achieving a score exceeding 0.99 in the training dataset. However, challenges arise when evaluating the model using classical metrics such as MSE, RMSE, MAE, MAPE, and  $R^2$ . These difficulties stem from the complex task of decoding the output quantum state  $\rho_{out}$  into classical data  $y_{out}$ . This situation underscores the intricate balance required in quantum-classical interfaces, where precision in decoding significantly influences the performance metrics. To address these issues, future enhancements may include investigating alternative decoding schemes or implementing error mitigation techniques. Additionally, optimization in DQNN poses challenges with current NISQ devices due to the necessity of extracting the quantum state from each layer. This complexity has shifted the focus towards developing and implementing PVQNN models.

### 2. PVQNN Performance and Challenges:

Analysis of PVQNN models reveals that they can be effectively trained with smaller datasets compared to traditional neural networks. Besides, the performance of these models is significantly influenced by the encoding methods employed, and the choice of ansatz structure. The effectiveness of a model relies not only on the performance of individual components but also on their synergistic integration. Besides, using data re-uploading has emerged as an important technique to enhance model performance, but the computational cost associated with increasing the number of hidden layers must be considered. Therefore, we should consider the whole picture when choosing a network structure. The best out-of-sample performance of our model, achieving an  $R^2$  of 0.99888, demonstrates the reliability of PVQNN in computing implied volatility.



# 6

## CONCLUSIONS AND FUTURE RESEARCH

### 6.1 CONCLUSIONS

In this thesis, our primary focus has been on the computation of implied volatility using two distinct architectures of quantum neural networks: DQNN and PVQNN.

Our analysis of DQNN highlighted its effective model fitting and capability to minimize loss functions. However, a key challenge with DQNN is its quantum state output, which requires a data decoding process. This process can lead to the loss of certain data features, adversely impacting the model's overall performance, and leading to a poor performance in the evaluation matrix. Therefore, selecting an appropriate data encoding method is crucial in this context. Additionally, DQNN's reliance on the output of network states at each layer poses implementation challenges on current quantum hardware, apart from that, training DQNN models on a classical simulator is time-consuming, and for these reasons, we have not conducted an in-depth study of this model.

In contrast, PVQNN showed more promise in addressing our research problem. With its classical data outputs, PVQNN allows for the application of traditional neural network optimization techniques. Our research looked into various aspects affecting PVQNN's performance, such as training data characteristics, the use of data re-uploading technology, network size, data encoding methods, quantum circuit construction, and the choice of observables. Through these investigations, we successfully developed a PVQNN model with an MSE of approximately  $1.5 \times 10^{-5}$  and an  $R^2$  around 0.999 in test sets on the classical simulator, demonstrating its efficacy in computing implied volatility.

The in-depth examination of the PVQNN model uncovered that the model required a relatively small dataset to achieve satisfied results, a characteristic that could be beneficial in practical applications where data availability is limited. Besides, the application of the data re-uploading technique greatly enhances model accuracy. This improvement is attributed to the property that PVQNN can be expressed in terms of partial Fourier series. Furthermore, the network's architecture, including encoding, hidden, and measurement layers, plays a vital role in its effectiveness. It suggests that each component of the network

does not operate in isolation but contributes to the overall model performance. Therefore, an integrated approach to network design and optimization is necessary.

Our experiments were primarily focused on assessing the model's expressive capabilities, rather than its training and computational efficiency. This focus stems from the fact that our simulations were conducted on classical computing systems. Consequently, the time metrics observed in our study were influenced by the limitations of classical simulations and do not reflect the performance in a true quantum computing environment. This distinction highlights the need for future research to evaluate the operational efficiency of these models on actual quantum hardware.

## 6.2 FUTURE RESEARCH

Future research in this field can be strategically directed towards three key areas to further advance our understanding and practical application of quantum neural networks in financial modeling:

1. **Advanced Exploration of DQNN:**

For DQNN, a deeper exploration of data encoding strategies is needed. This includes developing methods to ensure that crucial data characteristics are preserved during the decoding process. Additionally, investigating the impact of various network architectures on DQNN's modeling capabilities could yield insights into optimizing its structure for better performance. Such research would help in fine-tuning DQNN for specific financial applications, potentially enhancing its effectiveness in complex computations like implied volatility.

2. **Understanding PVQNN's Generalization Ability:**

Another promising research direction is to provide a deep analysis of the strong generalization ability observed in PVQNN. This entails a comprehensive examination of how different aspects of the model's architecture contribute to its overall performance. By studying the underlying mechanisms of PVQNN's generalization capabilities, future research can guide the development of more robust and efficient quantum neural network models, tailored to manage a broader range of financial modeling challenges.

3. **Testing on NISQ Devices:**

Perhaps the most critical avenue for future exploration is the practical application of PVQNN models on NISQ devices. Testing these models in a real quantum computing environment will provide valuable insights into their performance under realistic conditions, including the effectiveness of various noise reduction and optimization techniques. Such empirical studies are critical to determine the practical feasibility and scalability of quantum neural networks, especially in real-world financial environments where speed, efficiency, and accuracy are critical. This will not only validate the findings from simulated environments but also pave the way for the deployment of quantum computing solutions in the financial industry.



---

# BIBLIOGRAPHY

## REFERENCES

- [1] Christian Bayer, Blanka Horvath, Aitor Muguruza, Benjamin Stemper, and Mehdi Tomas. On deep calibration of (rough) stochastic volatility models, 2019.
- [2] Andres Hernandez. Model calibration with neural networks. *Available at SSRN 2812140*, 2016.
- [3] Shuaiqiang Liu, Anastasia Borovykh, Lech A. Grzelak, and Cornelis W. Oosterlee. A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*, 9(1), September 2019.
- [4] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 19(8):1271–1291, 2019.
- [5] Johannes Ruf and Weiguan Wang. Neural networks for option pricing and hedging: a literature review. *arXiv preprint arXiv:1911.05620*, 2019.
- [6] Christa Cuchiero, Wahid Khosrawi, and Josef Teichmann. A generative adversarial network approach to calibration of local stochastic volatility models. *Risks*, 8(4):101, September 2020.
- [7] Shuaiqiang Liu, Cornelis Oosterlee, and Sander Bohte. Pricing options and computing implied volatilities using neural networks. *Risks*, 7(1):16, February 2019.
- [8] Matthias Möller and Cornelis Vuik. On the impact of quantum computing technology on future developments in high-performance scientific computing. *Ethics and Information Technology*, 19(4):253–269, August 2017.
- [9] Nikitas Stamatopoulos, Daniel J. Egger, Yue Sun, Christa Zoufal, Raban Iten, Ning Shen, and Stefan Woerner. Option pricing using quantum computers. *Quantum*, 4:291, July 2020.
- [10] Daniel J. Egger, Claudio Gambella, Jakub Marecek, Scott McFaddin, Martin Mevissen, Rudy Raymond, Andrea Simonetto, Stefan Woerner, and Elena Yndurain. Quantum computing for finance: State-of-the-art and future prospects. *IEEE Transactions on Quantum Engineering*, 1:1–24, 2020.
- [11] Dylan Herman, Cody Googin, Xiaoyuan Liu, Alexey Galda, Ilya Safro, Yue Sun, Marco Pistoia, and Yuri Alexeev. A survey of quantum computing for finance. *arXiv preprint arXiv:2201.02773*, 2022.
- [12] Kerstin Beer. Quantum neural networks. *arXiv preprint arXiv:2205.08154*, 2022.

- [13] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, November 2019.
- [14] Min-Gang Zhou, Zhi-Ping Liu, Hua-Lei Yin, Chen-Long Li, Tong-Kai Xu, and Zeng-Bing Chen. Quantum neural network for quantum neural computing. *Research*, 6:0134, 2023.
- [15] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, August 2019.
- [16] ShiJie Wei, YanHu Chen, ZengRong Zhou, and GuiLu Long. A quantum convolutional neural network on nisq devices. *AAPPS Bulletin*, 32:1–11, 2022.
- [17] Oleksandr Kyriienko, Annie E. Paine, and Vincent E. Elfving. Solving nonlinear differential equations with differentiable quantum circuits. *Physical Review A*, 103(5), May 2021.
- [18] Asel Saginalieva, Mohammad Kordzanganeh, Nurbolat Kenbayev, Daria Kosichkina, Tatiana Tomashuk, and Alexey Melnikov. Hybrid quantum neural network for drug response prediction. *Cancers*, 15(10):2705, May 2023.
- [19] Takayuki Sakuma. Application of deep quantum neural networks to finance. *arXiv preprint arXiv:2011.07319*, 2020.
- [20] Eric Paquet and Farzan Soleymani. Quantumleap: Hybrid quantum neural network for financial predictions. *Expert Systems with Applications*, 195:116583, 2022.
- [21] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.
- [22] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.
- [23] E. Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Phys. Rev.*, 28:1049–1070, Dec 1926.
- [24] C.P. Williams. *Explorations in Quantum Computing*. Texts in Computer Science. Springer London, 2010.
- [25] Kerstin Beer, Daniel List, Gabriel Müller, Tobias J Osborne, and Christian Struckmann. Training quantum neural networks on nisq devices. *arXiv preprint arXiv:2104.06081*, 2021.
- [26] Gavin E Crooks. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. *arXiv preprint arXiv:1905.13311*, 2019.
- [27] Jun Zhang, Jiri Vala, Shankar Sastry, and K. Birgitta Whaley. Optimal quantum circuit synthesis from controlled-unitary gates. *Physical Review A*, 69(4), apr 2004.

- [28] Paolo Zanardi, Christof Zalka, and Lara Faoro. Entangling power of quantum evolutions. *Physical Review A*, 62(3), August 2000.
- [29] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii. Quantum circuit learning. *Physical Review A*, 98(3), September 2018.
- [30] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), mar 2019.
- [31] Yeong-Cherng Liang, Yu-Hao Yeh, Paulo E M F Mendonça, Run Yan Teh, Margaret D Reid, and Peter D Drummond. Quantum fidelity measures for mixed states. *Reports on Progress in Physics*, 82(7):076001, June 2019.
- [32] A. Uhlmann. The “transition probability” in the state space of a  $\ast$ -algebra. *Reports on Mathematical Physics*, 9(2):273–279, 1976.
- [33] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.
- [34] A. Bohm, P. Kielanowski, and G.B. Mainland. *Quantum Physics: States, Observables and Their Time Evolution*. Springer Netherlands, 2019.
- [35] Maria Schuld and Francesco Petruccione. *Machine learning with quantum computers*. Springer, 2021.
- [36] Maria Schuld. Supervised quantum machine learning models are kernel methods. *arXiv preprint arXiv:2101.11020*, 2021.
- [37] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I. Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, February 2020.
- [38] Francisco Javier Gil Vidal and Dirk Oliver Theis. Input redundancy for parameterized quantum circuits. *Frontiers in Physics*, 8:297, 2020.
- [39] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A*, 103(3), March 2021.
- [40] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, Shan You, and Dacheng Tao. Learnability of quantum neural networks. *PRX Quantum*, 2:040337, Nov 2021.
- [41] Charles Moussa, Jan N. van Rijn, Thomas Bäck, and Vedran Dunjko. *Hyperparameter Importance of Quantum Neural Networks Across Small Datasets*, page 32–46. Springer Nature Switzerland, 2022.
- [42] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, June 2021.

- 
- [43] Thomas Hubregtsen, Josef Pichlmeier, Patrick Stecher, and Koen Bertels. Evaluation of parameterized quantum circuits: on the relation between classification accuracy, expressibility, and entangling capability. *Quantum Machine Intelligence*, 3:1–19, 2021.
  - [44] Kerstin Beer, Dmytro Bondarenko, Terry Farrelly, Tobias J. Osborne, Robert Salzmann, Daniel Scheiermann, and Ramona Wolf. Training deep quantum neural networks. *Nature Communications*, 11, 2019.
  - [45] Matthias C. Caro, Hsin-Yuan Huang, M. Cerezo, Kunal Sharma, Andrew Sornborger, Lukasz Cincio, and Patrick J. Coles. Generalization in quantum machine learning from few training data. *Nature Communications*, 13(1), August 2022.
  - [46] Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Physical Review A*, 102(3), September 2020.
  - [47] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, March 2019.
  - [48] Sukin Sim, Peter D. Johnson, and Alán Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12), October 2019.

# A

## APPENDIX A

### A.1 DATA SIZE

Table A.1: Out-of-Sample PVQNN performance on different training data sizes.

Data Size	MSE	$R^2$
4	0.004858	0.638954
8	0.001168	0.913144
12	0.000956	0.928929
16	0.000627	0.953397
20	0.000473	0.964838
40	0.000571	0.957552
100	0.000558	0.95846
200	0.000507	0.96228
400	0.000452	0.96637
800	0.000391	0.97091
1600	0.000373	0.97221
4000	0.000362	0.97306
8000	0.000347	0.97419
16000	0.00035	0.97397

We expanded the range of the moneyiness  $S/K$ , from  $[0.95, 1.05]$  to a broader interval of  $[0.9, 1.1]$ , and then repeated the experiments. This was done to assess the universality of our conclusions. The results are as follows,

As it shows, the training results follow the same pattern as before, even on a wider range of data. The results for 160 training data are similar to the results for 800 training data.

Table A.2: Out-of-Sample PVQNN performance on different training data sizes in a wider range dataset.

Data Size	MSE	$R^2$
40	0.002112	0.84232
80	0.002011	0.84984
160	0.001733	0.87054
400	0.001695	0.87338
800	0.001587	0.88146

## A.2 DATA ENCODING

Table A.3: Out-of-Sample PVQNN performance with different data encoding methods.

Data Encoding	MSE	$R^2$
$e_1$	0.001139	0.91578
$e_2$	0.001139	0.91577
$e_3$	0.01357	$-1.374 \times 10^{-5}$
$e_5$	0.223	-15.44
$e_{10}$	0.00802	0.409
$e_6$	0.001138	0.91612
$e_7$	0.01357	$-2.387 \times 10^{-5}$
$e_8$	0.0007478	0.94490
$e_9$	0.06872	-4.063

## A.3 ANSATZ STRUCTURE

Table A.4: Out-of-Sample PVQNN performance across different ansatz structures

Ansatz Structure	Parameters	G	MSE	$R^2$
$h_1$	24	12	0.01412	0.01977
$h_2$	24	7	0.01413	0.01856
$h_3$	24	0	$7.212 \times 10^{-4}$	0.94992
$h_4$	24	0	0.00111	0.92259
$h_5$	24	0	$7.443 \times 10^{-4}$	0.94831
$h_6$	24	0	$7.428 \times 10^{-4}$	0.94841
$h_7$	24	2	$6.608 \times 10^{-4}$	0.95411
$h_8$	24	1	$8.761 \times 10^{-4}$	0.93916
$h_9$	24	4	$5.357 \times 10^{-4}$	0.96280
$h_{10} (e_1)$	24	12	0.00413	0.71299
$h_{10} (e_2)$	24	24	-	-
$h_{11}$	16	0	$6.779 \times 10^{-4}$	0.95293
$h_{12}$	16	4	$8.720 \times 10^{-4}$	0.93945
$h_{13}$	28	4	$8.969 \times 10^{-4}$	0.93772
$h_{14}$	16	0	0.00125	0.91306
$h_{15}$	20	5	$8.424 \times 10^{-4}$	0.94151

## A.4 LAGER DATASET

We expanded the range of the key parameter in our problem, the moneyiness  $S/K$ , from  $[0.98, 1.02]$  to a broader interval of  $[0.9, 1.1]$ . This was done to assess the adaptability of our model to a wider range of scenarios. For this purpose, we employed the third model, which previously showed the best performance. This model was trained on 160 data points with a batch size of 64 over 200 epochs, keeping other parameters consistent with the initial setup.

However, the expanded test presented challenges. The out-of-sample performance of the model, particularly the  $R^2$  value, dropped to around 0.89. This was a significant deviation from the earlier high performance. One potential solution to improve the model's performance on this broader data range could be to increase the number of hidden layers or the size of the training dataset. However, such adjustments would demand substantially more computational resources and time, which might not be feasible or efficient.

To address these challenges, we proposed another approach: dividing the larger interval into smaller sub-intervals and training separate models for each. This strategy aimed to mitigate the issues of poor training results and extended training time associated with the larger data range. Models A, B, and C were structured identically to Model 3 and were each tested on a dataset of 5000 points, distributed as the training data.

The results of this segmented approach, as detailed in Table A.5, indicate positive outcomes. The method of dividing the data into smaller intervals appears to be effective for the problem at hand. By training separate models on smaller intervals, we were able to maintain high accuracy and reliability in predictions across a broader range of moneyiness values. This strategy not only preserved the high  $R^2$  values, around 0.999 but also ensured efficiency in training. These outcomes suggest that this approach offers a feasible solution

to the challenges encountered due to the expanded data range. The effectiveness of this method highlights the potential benefits of adaptable and innovative strategies in model training, especially in the context of complex and diverse data sets.

Table A.5: Out-of-Sample performance of PVQNN models on extended Moneyness range dataset

Model	Moneyness $S/K$	Training Data Size	Batch Size	Epochs	MSE	RMSE	MAE	MAPE	$R^2$
A	[0.9,1.1]	160	64	200	0.0014897	0.038597	0.031877	0.067713	0.88899
B	[0.9,0.95]	80	20	500	$1.750 \times 10^{-5}$	0.00418	0.00346	0.00726	0.99870
C	[1.05,1.1]	80	20	500	$1.097 \times 10^{-5}$	0.00331	0.00222	0.00466	0.99918