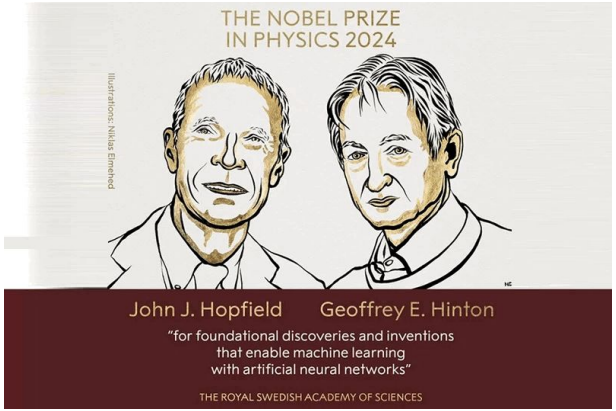
Machine learning for Quantum Key Distribution network optimization

Nonlinear Quantum Photonics

NQCIS weden

Master Project Course SK2002

Master Student: Shek Lun Leung, Alan
Supervisor: Erik Svanberg
Co-Supervisors: Giulio Foletto
Vaishali Adya
Examiner: Katia Gallo



KTH VETENSKAP OCH KONST

Overall Workflow of the project

01

$$l = s_{X,0} + s_{X,1} - s_{X,1} h(\phi_X) - \lambda_{EC} - 6\log_2\left(\frac{2}{\epsilon_{\text{sec}}}\right) - \log_2\left(\frac{2}{\epsilon_{\text{hash}}}\right)$$

$$\nu_{Z,1} \leq \tau_1 \frac{m^2_{Z,\mu_2} - m^2_{Z,\mu_3}}{\mu_2 - \mu_3},$$

Where:

$$m^2_{Z,k} = \frac{e^k}{p_k}\left[m_{Z,k} \pm \sqrt{\frac{m_Z}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad \forall k \in \mathcal{K},$$

And:

$$m_Z = \sum_{k\in\mathcal{K}} m_{Z,k}.$$

The phase error rate of single-photon events in $\mathbf{X}_A$ is [11]:

$$\phi_X = \frac{c_{X,1}}{s_{X,1}} \leq \frac{\nu_{Z,1}}{s_{Z,1}} + \gamma\left(\epsilon_{\text{sec}}, \frac{\nu_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1}\right),$$

Where:

$$\gamma(a,b,c,d) = \sqrt{\frac{(c+d)(1-b)b}{cd\ln 2}\log_2\left(\frac{c+d}{cd(1-b)b}\frac{21^2}{a^2}\right)}.$$

The key rate becomes:

$$R = \frac{l}{N}.$$

Introduction and Motivation on building the script from equations found in the literature to verify the key rate in BB84 Protocol

Overall Workflow of the project

Overall Workflow of the project

01

$$l = s_{X,0} + s_{X,1} - s_{X,1}h(\phi_X) - \lambda_{EC} - 6\log_2\left(\frac{2}{\epsilon_{\text{sec}}}\right) - \log_2\left(\frac{2}{\epsilon_{\text{hash}}}\right)$$

$$\nu_{Z,1} \leq \tau_1 \frac{m^L_{Z,\mu_2} - m^U_{Z,\mu_3}}{\mu_2 - \mu_3},$$

Where:

$$m^L_{Z,k} = \frac{e^k}{p_k}\left[m_{Z,k} \pm \sqrt{\frac{m_Z}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad \forall k \in \mathcal{K},$$

And:

$$m_Z = \sum_{k\in\mathcal{K}} m_{Z,k}.$$

The phase error rate of single-photon events in $\mathbf{X}_A$ is [11]:

$$\phi_X = \frac{c_{X,1}}{s_{X,1}} \leq \frac{\nu_{Z,1}}{s_{Z,1}} + \gamma\left(\epsilon_{\text{sec}}, \frac{\nu_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1}\right),$$

Where:

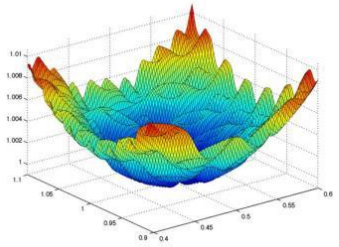
$$\gamma(a,b,c,d) = \sqrt{\frac{(c+d)(1-b)b}{cd\ln 2}\log_2\left(\frac{c+d}{cd(1-b)b}\frac{21^2}{a^2}\right)}.$$

The key rate becomes:

$$R = \frac{l}{N}.$$

Introduction and Motivation on building the script from equations found in the literature to verify the key rate in BB84 Protocol

02



Optimization with Dual Annealing to generate dataset of 1,000 data point of optimized parameters with key rate (~ 45 mins) for each received photon number

Overall Workflow of the project

01

$$l = s_{X,0} + s_{X,1} - s_{X,1}h(\phi_X) - \lambda_{EC} - 6\log_2\left(\frac{2}{\epsilon_{\text{sec}}}\right) - \log_2\left(\frac{2}{\epsilon_{\text{hash}}}\right)$$

$$\nu_{Z,1} \leq \tau_1 \frac{m^z_{Z,\mu_2} - m^z_{Z,\mu_3}}{\mu_2 - \mu_3},$$

Where:

$$m^z_{Z,k} = \frac{e^k}{p_k}\left[m_{Z,k} \pm \sqrt{\frac{m_Z}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad \forall k \in \mathcal{K},$$

And:

$$m_Z = \sum_{k\in\mathcal{K}} m_{Z,k}.$$

The phase error rate of single-photon events in $\mathbf{X}_A$ is [11]:

$$\phi_X = \frac{c_{X,1}}{s_{X,1}} \leq \frac{\nu_{Z,1}}{s_{Z,1}} + \gamma\left(\epsilon_{\text{sec}}, \frac{\nu_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1}\right),$$

Where:

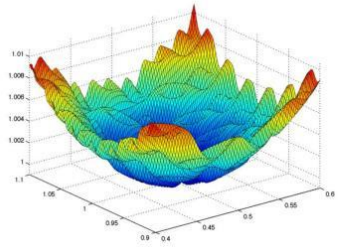
$$\gamma(a,b,c,d) = \sqrt{\frac{(c+d)(1-b)b}{cd\ln 2}\log_2\left(\frac{c+d}{cd(1-b)b}\frac{21^2}{a^2}\right)}.$$

The key rate becomes:

$$R = \frac{l}{N}.$$

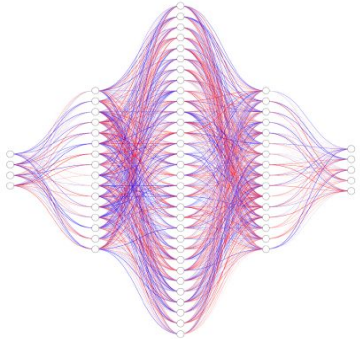
Introduction and Motivation on building the script from equations found in the literature to verify the key rate in BB84 Protocol

02



Optimization with Dual Annealing to generate dataset of 1,000 data point of optimized parameters with key rate for each received photon number

03



Train the neural network model from the optimized parameters in the dataset for 5,000 epoch

Overall Workflow of the project

01

$$l = s_{X,0} + s_{X,1} - s_{X,1}h(\phi_X) - \lambda_{EC} - 6\log_2\left(\frac{2}{\epsilon_{\text{sec}}}\right) - \log_2\left(\frac{2}{\epsilon_{\text{hash}}}\right)$$

$$\nu_{Z,1} \leq \tau_1 \frac{m^z_{Z,\mu_2} - m^z_{Z,\mu_3}}{\mu_2 - \mu_3},$$

Where:

$$m^z_{Z,k} = \frac{e^k}{p_k}\left[m_{Z,k} \pm \sqrt{\frac{m_Z}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad \forall k \in \mathcal{K},$$

And:

$$m_Z = \sum_{k\in\mathcal{K}} m_{Z,k}.$$

The phase error rate of single-photon events in $\mathbf{X}_A$ is [11]:

$$\phi_X = \frac{c_{X,1}}{s_{X,1}} \leq \frac{\nu_{Z,1}}{s_{Z,1}} + \gamma\left(\epsilon_{\text{sec}}, \frac{\nu_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1}\right),$$

Where:

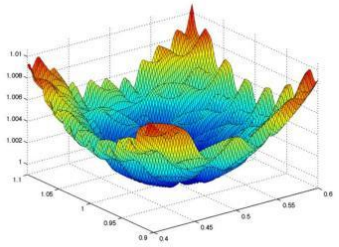
$$\gamma(a,b,c,d) = \sqrt{\frac{(c+d)(1-b)b}{cd\ln 2}\log_2\left(\frac{c+d}{cd(1-b)b}\frac{21^2}{a^2}\right)}.$$

The key rate becomes:

$$R = \frac{l}{N}.$$

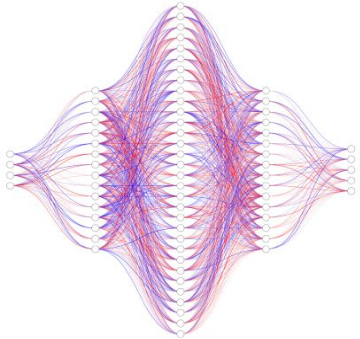
Introduction and Motivation on building the script from equations found in the literature to verify the key rate in BB84 Protocol

02



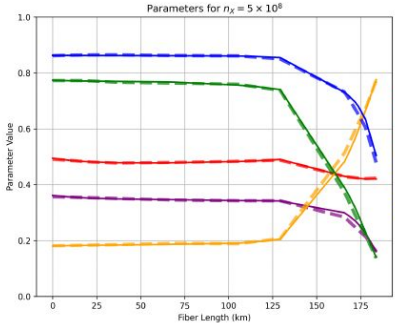
Optimization with Dual Annealing to generate dataset of 1,000 data point of optimized parameters with key rate (~ 45 mins) for each received photon number

03



Train the neural network model from the optimized parameters in the dataset for 5,000 epochs (< 6 mins)

04



Import 100 points of unseen dataset to the model to obtain the inference (~ 10s for 1,000 data points)

KTH VETENSKAP OCH KONST

01

$$l = s_{X,0} + s_{X,1} - s_{X,1}h(\phi_X) - \lambda_{EC} - 6\log_2\left(\frac{2}{\epsilon_{\text{sec}}}\right) - \log_2\left(\frac{2}{\epsilon_{\text{hash}}}\right)$$

$$\nu_{Z,1} \leq \tau_1 \frac{m^z_{Z,\mu_2} - m^z_{Z,\mu_3}}{\mu_2 - \mu_3},$$

Where:

$$m^z_{Z,k} = \frac{e^k}{p_k}\left[m_{Z,k} \pm \sqrt{\frac{m_Z}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad \forall k \in \mathcal{K},$$

And:

$$m_Z = \sum_{k\in\mathcal{K}} m_{Z,k}.$$

The phase error rate of single-photon events in $\mathbf{X}_A$ is [11]:

$$\phi_X = \frac{c_{X,1}}{s_{X,1}} \leq \frac{\nu_{Z,1}}{s_{Z,1}} + \gamma\left(\epsilon_{\text{sec}}, \frac{\nu_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1}\right),$$

Where:

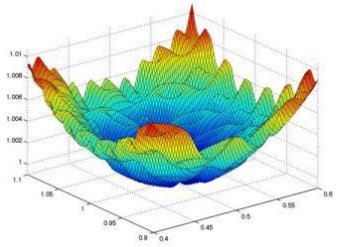
$$\gamma(a,b,c,d) = \sqrt{\frac{(c+d)(1-b)b}{cd\ln 2}\log_2\left(\frac{c+d}{cd(1-b)b}\frac{21^2}{a^2}\right)}.$$

The key rate becomes:

$$R = \frac{l}{N}.$$

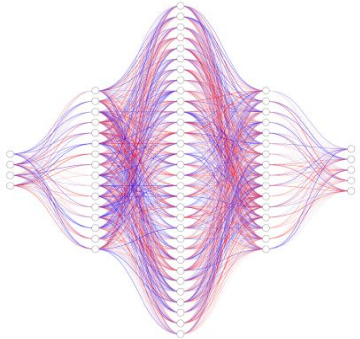
Introduction and Motivation on building the script from equations found in the literature to verify the key rate in BB84 Protocol

02



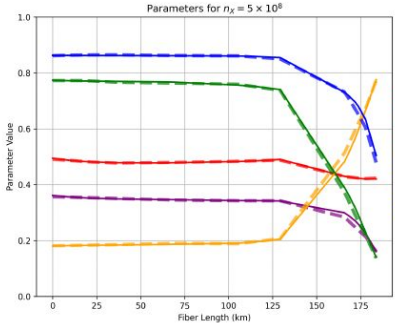
Optimization with Dual Annealing to generate dataset of 1,000 data point of optimized parameters with key rate (~ 45 mins) for each received photon number

03



Train the neural network model from the optimized parameters in the dataset for 5,000 epochs (< 6 mins)

04



Import 100 points of unseen dataset to the model to obtain the inference (~ 10s for 1,000 data points)

Overall Workflow of the project

Background

Method

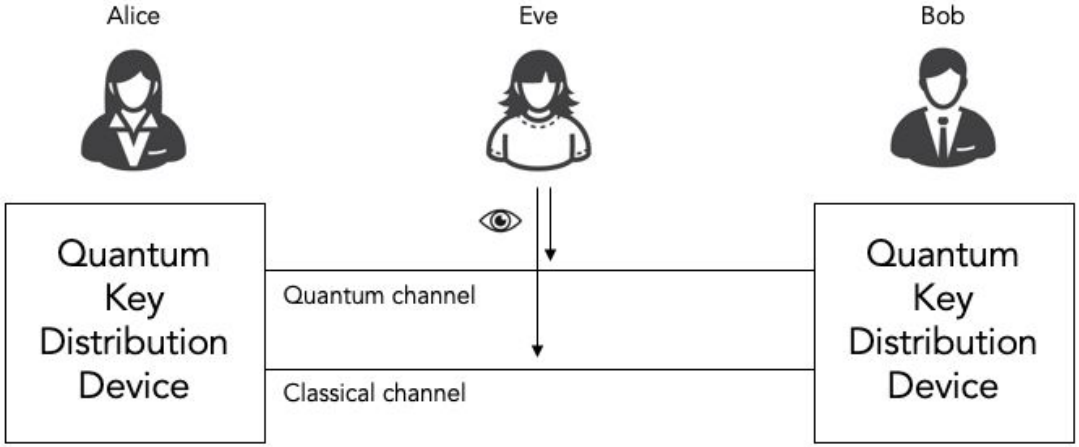
Result

KTH VETENSKAP OCH KONST

Workflow 01

Introduction and Motivation on building the script from equations found in the literature to verify the key rate in BB84 Protocol

Background on Quantum Key Distribution



QKD for secure communication over quantum networks

Real-World QKD System

1. Limited Transmission Time
2. System Imperfections

=> Optimization in Parameters

=> Optimization in key rates

Maximizing key rates -> tuning for Optimizing Parameters

1. Laser Pulse Signal Intensities
2. Basis Measurement Choices
3. Corresponding Probabilities

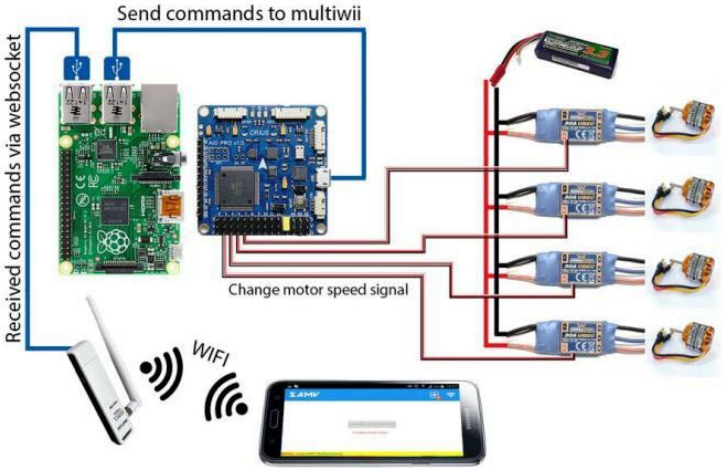
QKD for Mobile Platforms, Quantum IoT and future Network

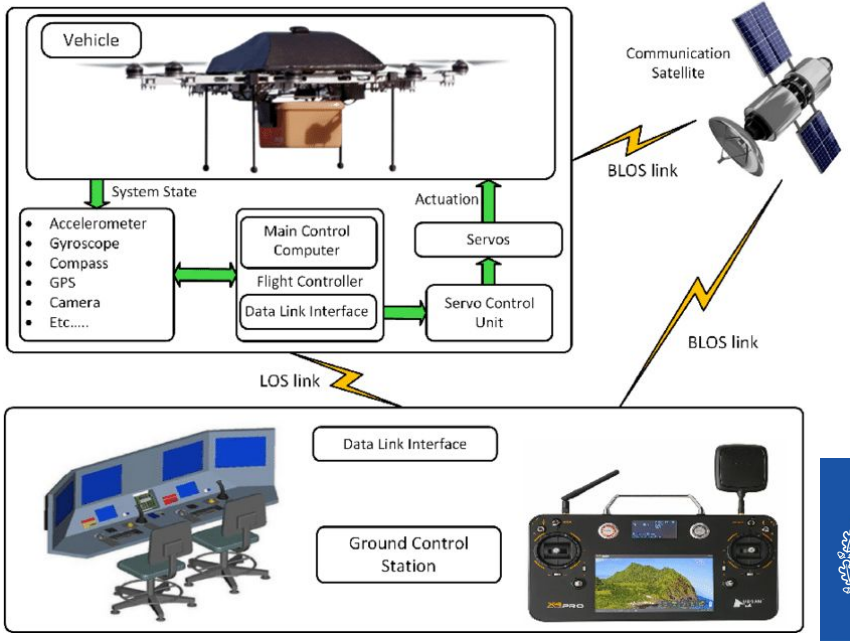
Traditional Optimization is computationally intensive!

Especially in resource-constrained platform

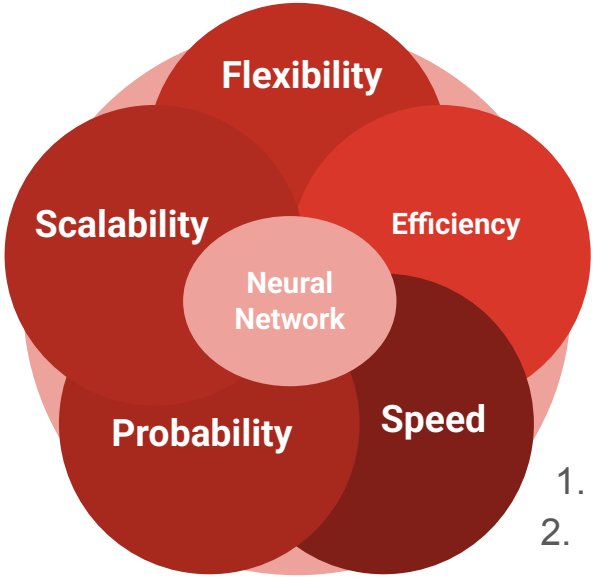
Low-power devices -> real-time optimization

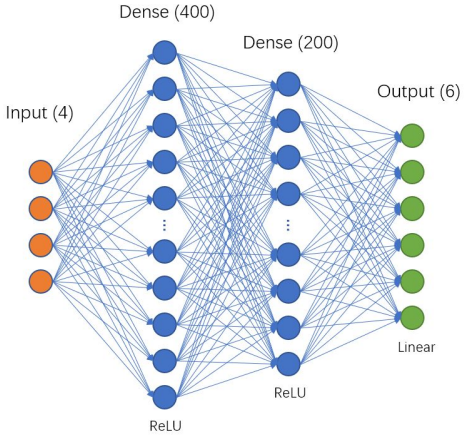
For dynamic change and parameter adjustment





Why Machine Learning? Optimal Solution for QKD





Applications

1. Free-Space QKD (Drone, Satellite, Handheld)
2. Large Scale Quantum Internet-of-Things (QIoT)

Protocol	Device	NN accelerator	Local search	NN	Power consumption
MDI-QKD	Desktop PC	Titan Xp GPU	0.1s	0.5-1.0ms	~350w
TF-QKD	Desktop PC	Titan Xp GPU	2s	0.5-1.0ms	~350w
MDI-QKD	iPhone XR	on-board neural engine	0.2s	~1ms	<5w
TF-QKD	iPhone XR	on-board neural engine	N/A	~1ms	<5w
MDI-QKD	Raspberry Pi 3	Intel neural compute stick	3-5s	2-3ms	<5w
TF-QKD	Raspberry Pi 3	Intel neural compute stick	15-16s	3ms	<5w

KTH VETENSKAP OCH KONST

01 - Build the script from equations in the literature to verify the key rate in BB84 Protocol

Literature Review the machine learning result from the use of Neural Network for fast and efficient parameter optimization in QKD protocols

Lim, C. C. W., Curty, M., Walenta, N., Xu, F., & Zbinden, H. (2014). Concise security bounds for practical decoy-state quantum key distribution. *Physical Review A*, *89*(2), 022307.

Wang, W., & Lo, H. K. (2019). Machine learning for optimal parameter prediction in quantum key distribution. *Physical Review A*, *100*(6), 062334.

$$l = s_{X,0} + s_{X,1} - s_{X,1}h(\phi_X) - \lambda_{EC} - 6\log_2\left(\frac{2}{\epsilon_{\text{sec}}}\right) - \log_2\left(\frac{2}{\epsilon_{\text{hash}}}\right)$$

$$\nu_{Z,1} \leq \tau_1 \frac{m^{+}_{Z,\mu_2} - m^{-}_{Z,\mu_3}}{\mu_2 - \mu_3},$$

Where:

$$m^{\pm}_{Z,k} = \frac{e^k}{p_k}\left[m_{Z,k} \pm \sqrt{\frac{m_Z}{2}\ln\frac{21}{\epsilon_{\text{sec}}}}\right], \quad \forall k \in \mathcal{K},$$

And:

$$m_Z = \sum_{k\in\mathcal{K}} m_{Z,k}.$$

The phase error rate of single-photon events in $\mathbf{X}_A$ is [11]:

$$\phi_X = \frac{c_{X,1}}{s_{X,1}} \leq \frac{\nu_{Z,1}}{s_{Z,1}} + \gamma\left(\epsilon_{\text{sec}}, \frac{\nu_{Z,1}}{s_{Z,1}}, s_{Z,1}, s_{X,1}\right),$$

Where:

$$\gamma(a,b,c,d) = \sqrt{\frac{(c+d)(1-b)b}{cd\ln 2}\log_2\left(\frac{c+d}{cd(1-b)b}\frac{21^2}{a^2}\right)}.$$

The key rate becomes:

$$R = \frac{l}{N},$$

$$\text{Input} : \vec{e} = [e_1, e_2, e_3, e_4] = [L, P_{dc}, e_d, n_X]$$

$$\text{Output} : \vec{p} = [p_1, p_2, p_3, p_4, p_5] = [\mu_1, \mu_2, P_{\mu_1}, P_{\mu_2}, P_X]$$

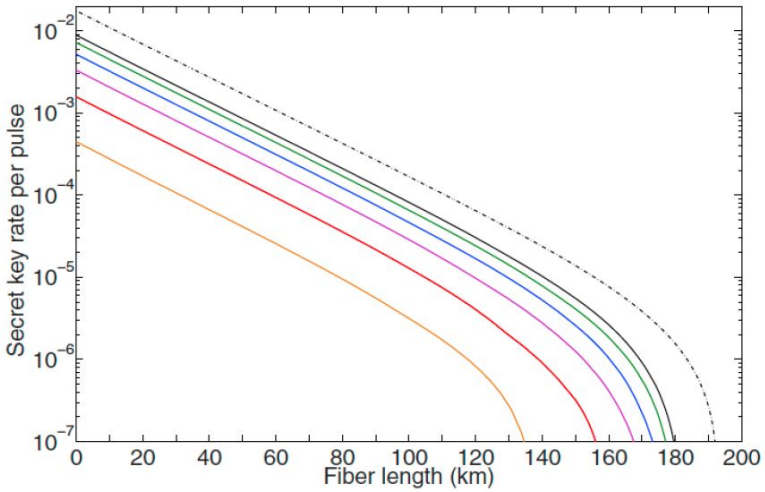
$$\vec{e} \rightarrow \vec{p}$$

$$\text{QKD key Rate} = Rate = R(\vec{e}, \vec{p})$$

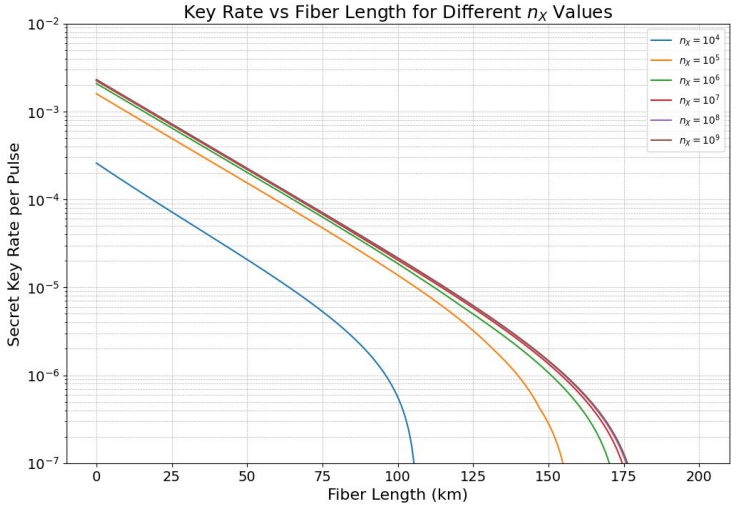
$$R_{max}(\vec{e}) = max_{\vec{p}\in P} R(\vec{e}, \vec{p})$$

$$\vec{p_{opt}}(\vec{e}) = argmax_{\vec{p}\in P} R(\vec{e}, \vec{p})$$

01 - Build the script from equations in the literature to verify the key rate in BB84 Protocol comparable to the literature



Optimization from Lim .et. al (2014)



Simulation from equation in Lim et. at (2014) with fixed values of parameters

n_X	100 km			150 km		
	Key Rate (R)	$\log_{10}(R)$	Gain (%) vs. Prev. n_X	Key Rate (R)	$\log_{10}(R)$	Gain (%) vs. Prev. n_X
10^4	5.53×10^{-7}	-6.26	N/A	< 0	N/A	N/A
10^5	1.37×10^{-5}	-4.86	2372.3	2.85×10^{-7}	-6.55	N/A
10^6	1.85×10^{-5}	-4.73	35.5	1.07×10^{-6}	-5.97	276.9
10^7	2.04×10^{-5}	-4.69	10.0	1.35×10^{-6}	-5.87	26.0
10^8	2.13×10^{-5}	-4.67	4.5	1.44×10^{-6}	-5.84	6.8
10^9	2.16×10^{-5}	-4.67	1.4	1.47×10^{-6}	-5.83	2.0

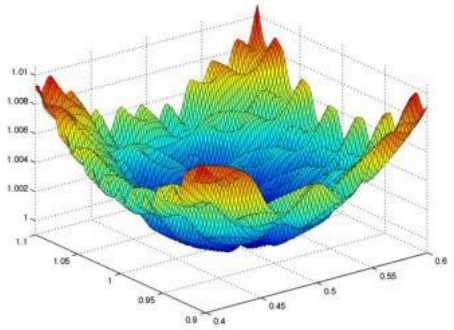
Table 4: Optimized secret key rates (R) and percentage gains achieved by increasing block size (n_X) at selected fiber lengths.

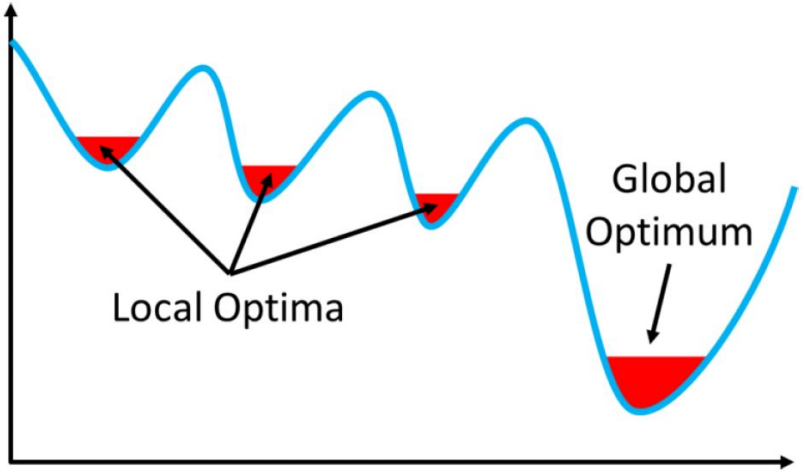
- Not optimized parameters
- Lower key rate values
- Similar trend across the distance

Workflow 02

Optimization with Dual Annealing to generate dataset of 1,000 data point of optimized parameters with key rate (~ 45 mins) for each received photon number

Background on Optimization



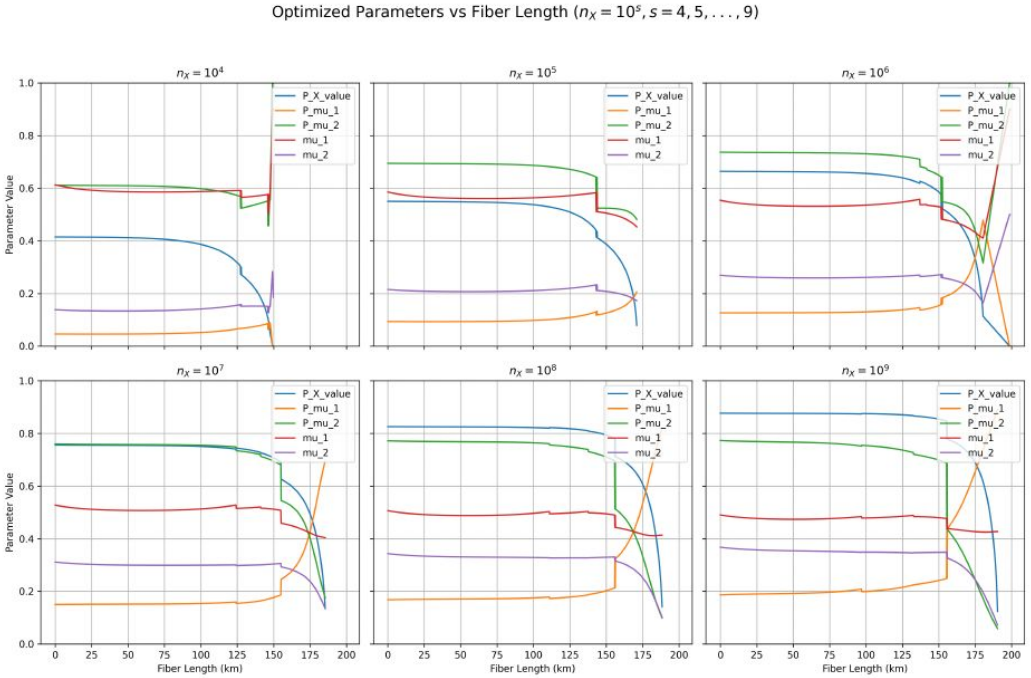


Optimization => maximum key rate

Feature	Local Search	Global Search
Search Scope	Local region	Entire parameter space
Efficiency	Fast, efficient	Slow, resource-intensive
Risk of Local Minima	High	Low
Applicability	Convex functions	Non-convex landscapes

Dual Annealing utilizes both search algorithms

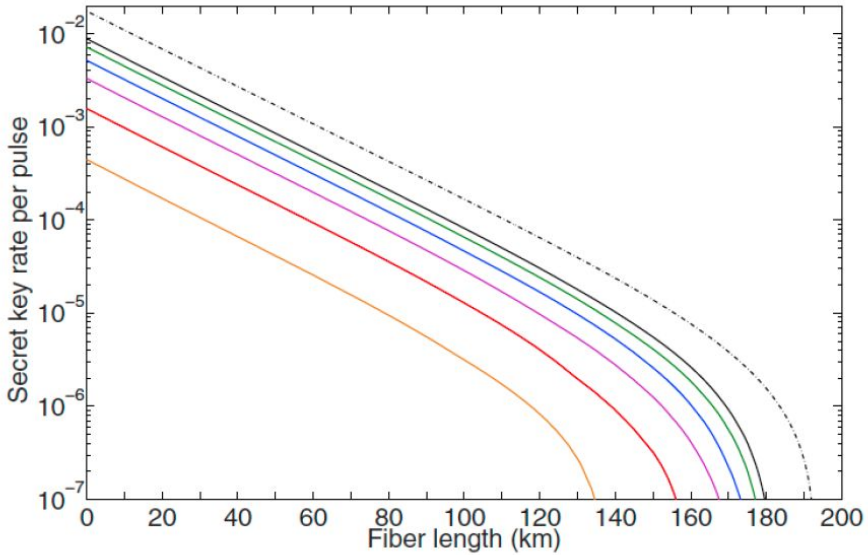
02 - Optimization with Dual Annealing to generate dataset of 1,000 data point of optimized parameters with key rate (~ 45 mins) for each received photon number



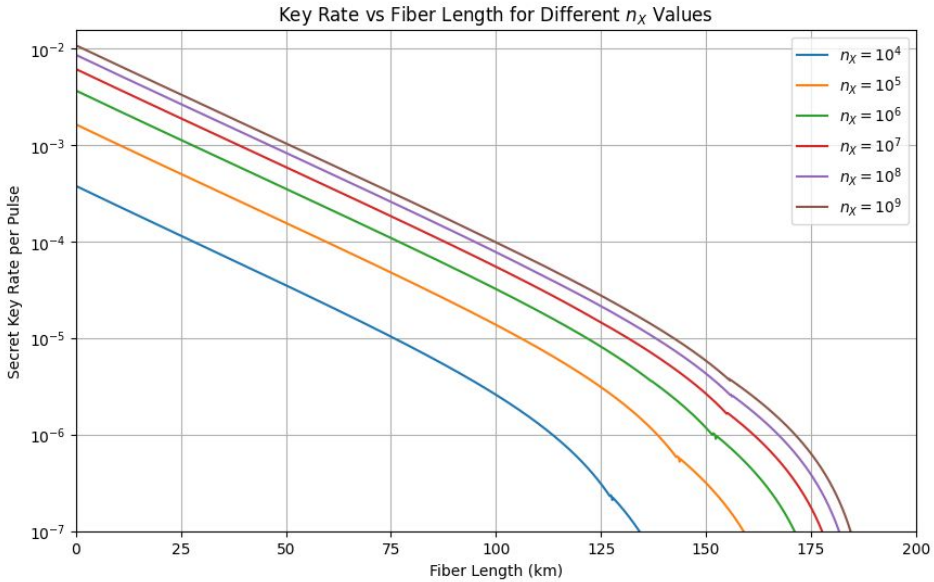
Block Size n_X	μ_1	μ_2	P_{μ_1}	P_{μ_2}	P_X
10^4	0.65	0.15	0.05	0.61	0.425
10^5	0.62	0.24	0.10	0.70	0.55
10^6	0.68	0.30	0.14	0.74	0.66
10^7	0.55	0.34	0.15	0.75	0.75
10^8	0.54	0.375	0.16	0.775	0.83
10^9	0.52	0.40	0.18	0.785	0.88

Table 2: Initial guess values for different block sizes n_X.

02 - Optimization with Dual Annealing to generate dataset of 1,000 data point of optimized parameters with key rate (~ 45 mins) for each received photon number



Optimization from Lim .et. al (2014)



Optimization from Dual Annealing, key rate calculated from optimized parameters

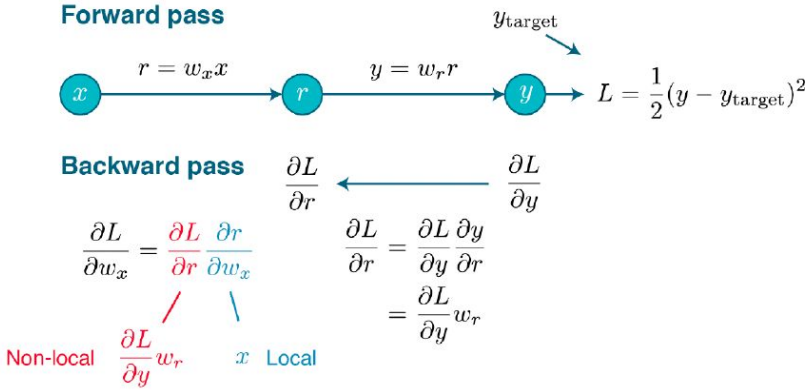
Workflow 03

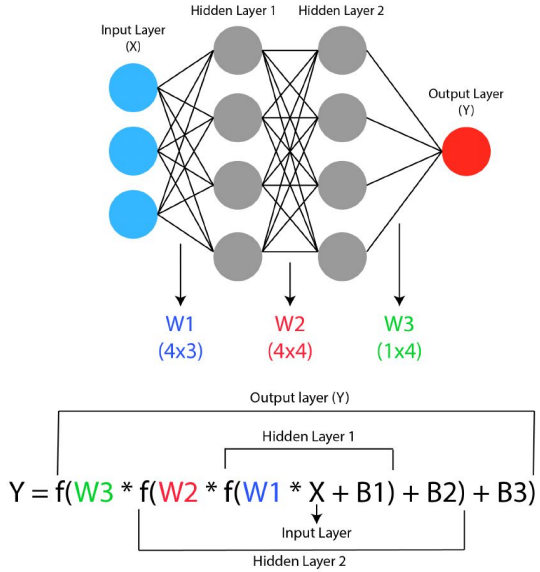
Train the neural network model from the optimized parameters in the dataset for 5,000 epochs (< 6 mins)

Background on Neural Network

Input -> Model (Forward Propagation)

Input data is passed through the layers of neural network, each of which performing a transformation to produce outputs





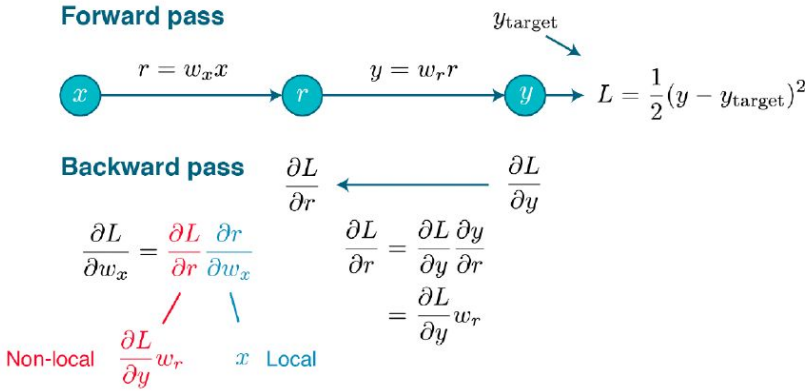
Background on Neural Network

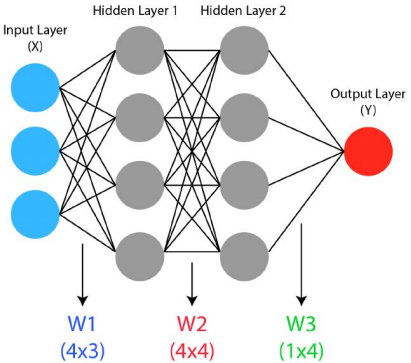
Input -> Model (Forward Propagation)

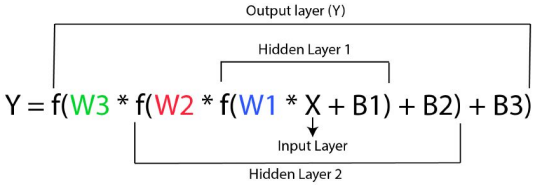
Prediction -> Loss Calculation

Input data is passed through the layers of neural network, each of which performing a transformation to produce outputs

Loss function is used to compute the difference between the predicted output and the actual target values.







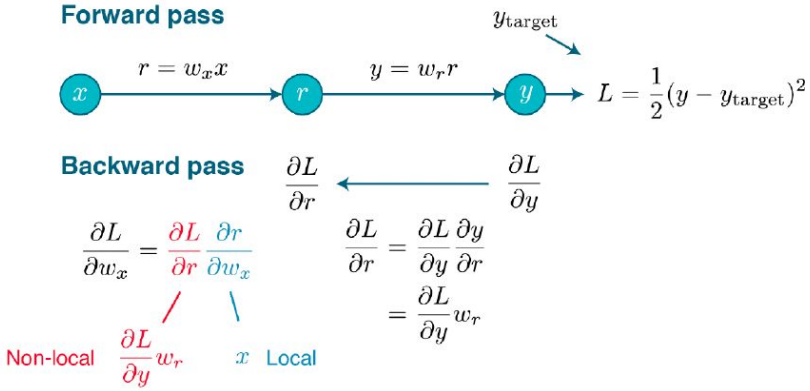
Background on Neural Network

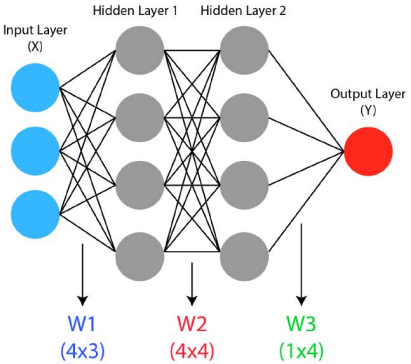
Input -> Model (Forward Propagation) | Prediction -> Loss Calculation | Back Propagation

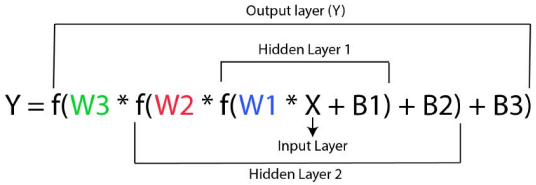
Input data is passed through the layers of neural network, each of which performing a transformation to produce outputs

Loss function is used to compute the difference between the predicted output and the actual target values.

The algorithm computes the loss gradients w.r.t the weight of each layers using the chain rule.







Background on Neural Network

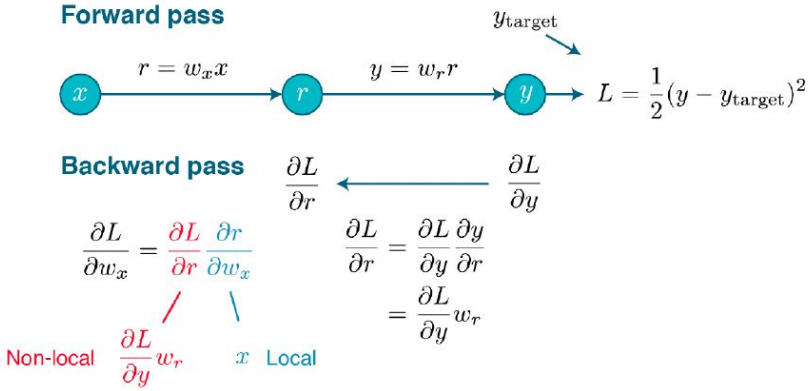
Input -> Model (Forward Propagation) | Prediction -> Loss Calculation | Back Propagation | Weight Update

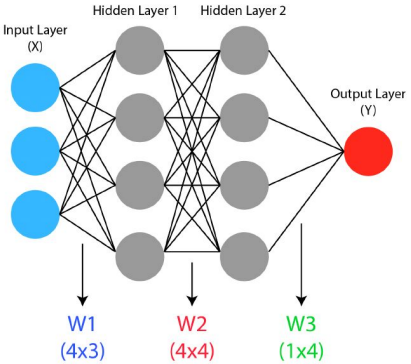
Input data is passed through the layers of neural network, each of which performing a transformation to produce outputs

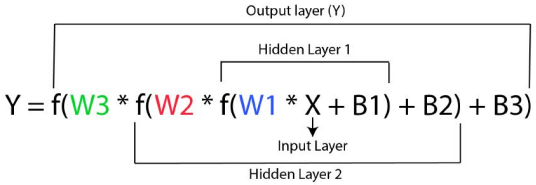
Loss function is used to compute the difference between the predicted output and the actual target values.

The algorithm computes the loss gradients w.r.t the weight of each layers using the chain rule.

Optimizer adjusts the weights based on the computed gradients and the learning rate.







Background on Neural Network

Input -> Model (Forward Propagation) | Prediction -> Loss Calculation | Back Propagation | Weight Update | Repeat

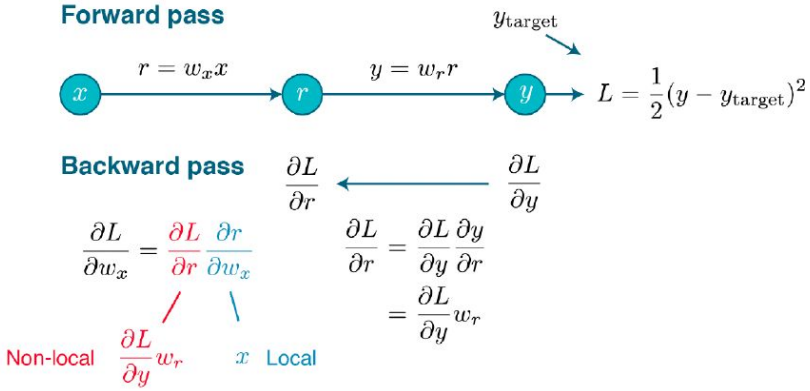
Input data is passed through the layers of neural network, each of which performing a transformation to produce outputs

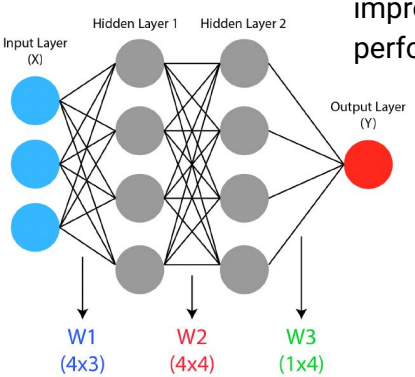
Loss function is used to compute the difference between the predicted output and the actual target values.

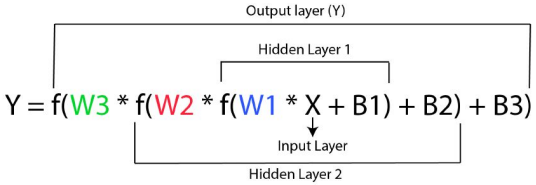
The algorithm computes the loss gradients w.r.t the weight of each layers using the chain rule.

Optimizer adjusts the weights based on the computed gradients and the learning rate.

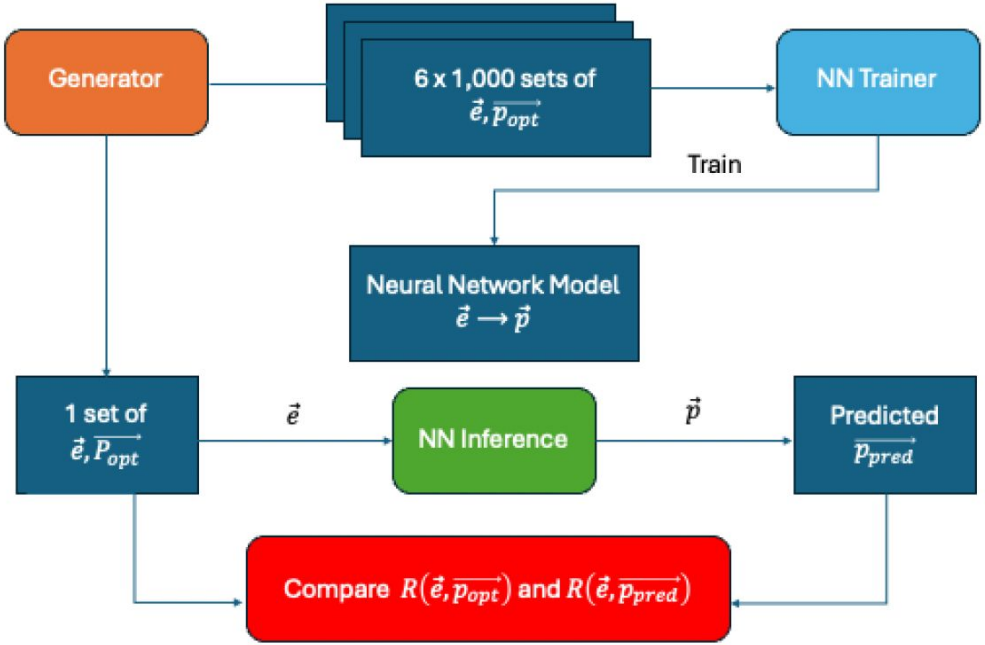
The step is repeated for each mini-batch of data during each epoch, progressively improving the model's performance.



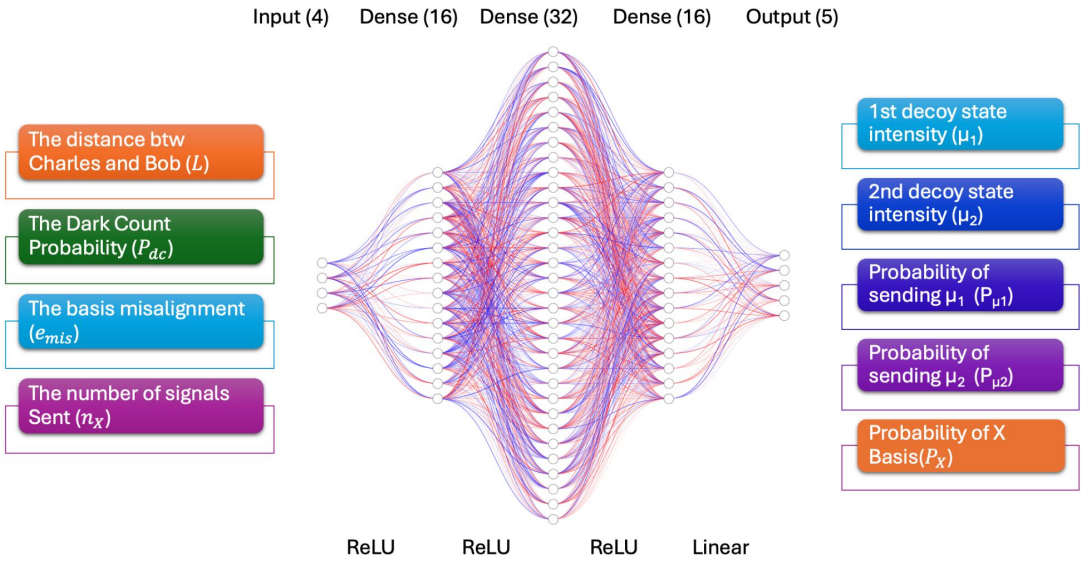


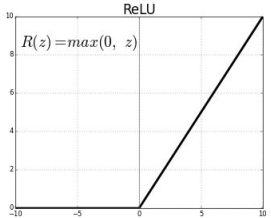


03 - Train the neural network model from the optimized parameters in the dataset for 5,000 epochs (< 6 mins)



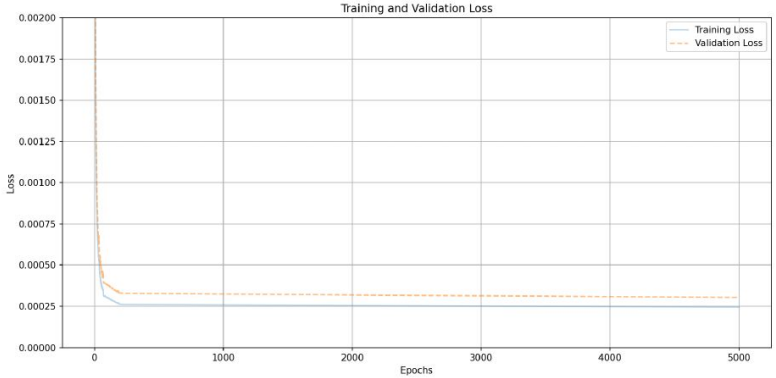
03 - Train the neural network model from the optimized parameters in the dataset for 5,000 epochs (< 6 mins)



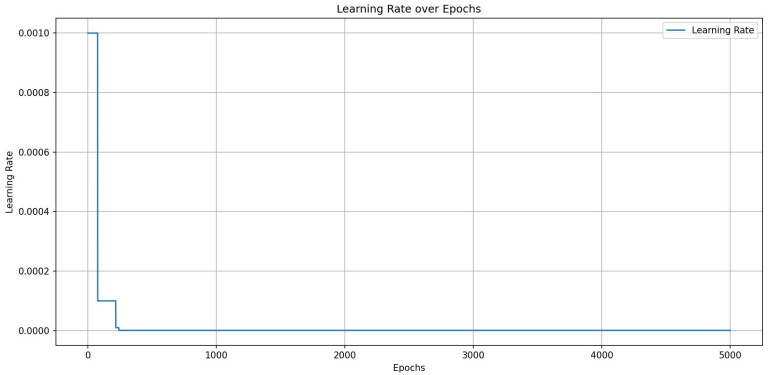


$$\text{ReLU} : max(0, z),\ z = \sum w_i x_i + b$$

03 - Import 100 points of unseen dataset to the model to obtain the inference (~ 10s for 1,000 data points)



- Rapid decrease in the first 100 epochs
- Model learn quickly the features and patterns from training data
- The rate of improvement slows
- After 300 epochs, minimal change in loss
- The model converge effectively
- No overfitting, good generalization
- Extended training ensure the stability

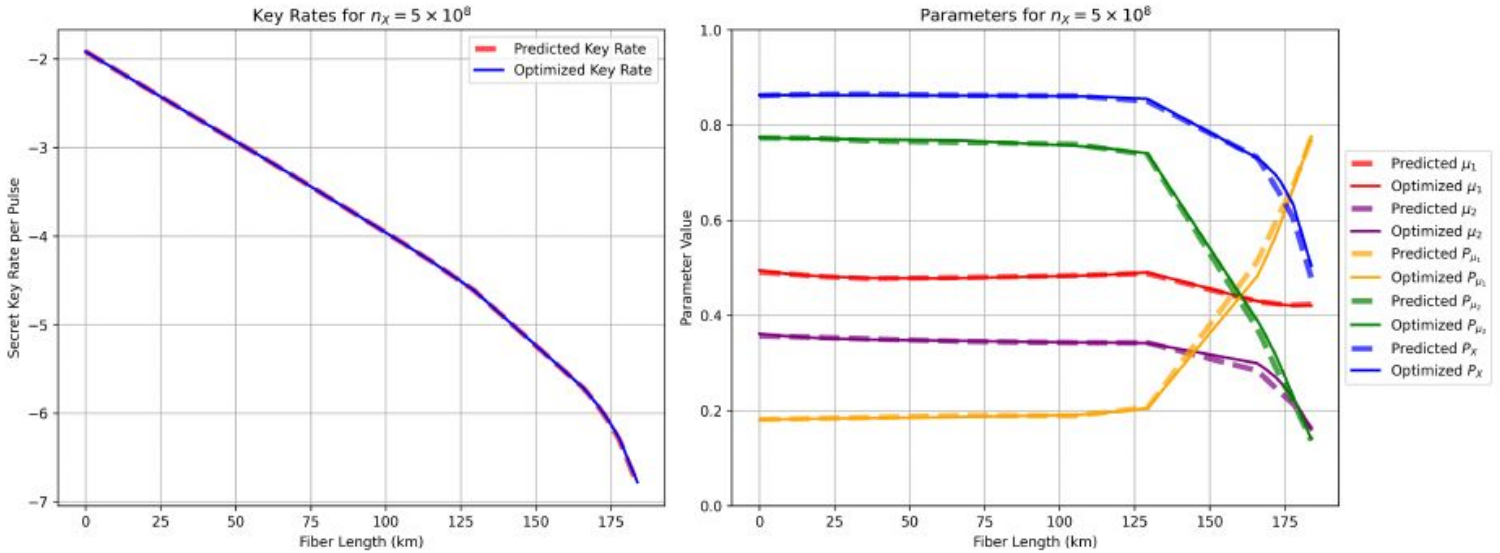


- The learning rate scheduler dynamically adjust the learning rate based on the loss performance
- Improvement in loss began to slow, triggering the reduction on learning rate
- Step-down signifies the loss plateaued for a number of epochs
- Ensuring model continues with small, stable updates, fine-tuning without the risk of divergence

Workflow 04

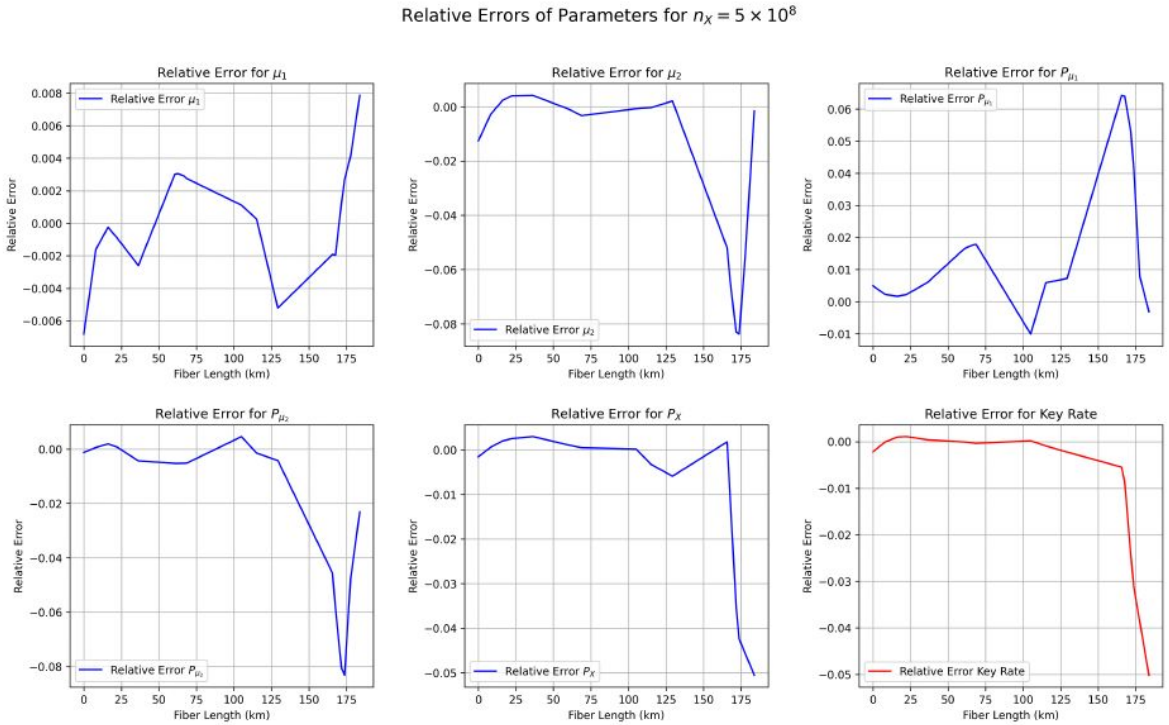
Import 100 points of unseen dataset to the model to obtain the inference (~ 10s for 1,000 data points)

04 - Import 100 points of unseen dataset to the model to obtain the inference (~ 10s for 1,000 data points)



Hardly distinguish the difference along the distance (0 km -125 km)

04 - Import 100 points of unseen dataset to the model to obtain the inference (~ 10s for 1,000 data points)



Relative Error < 0.1 = 10%

Computational Efficiency and Practical Considerations

Aspect	Details (BB84 Protocol)
Evaluation Platform	MacBook Pro M2 Pro (12-core CPU, 19-unit GPU)
Optimization Method	Dual Annealing (SciPy implementation)
Optimization Performance	CPU-bound (10 cores used); approx. 45 min per 1,000 data points
Neural Network Training	GPU-accelerated (19-unit GPU); approx. 5 min 30 s (5,000 epochs on 6,000 points)
Neural Network Inference	GPU-accelerated; approx. 10 s per 1,000 data points

45 mins = 2,700 seconds for 1,000 points via DA
270-fold speed up

Acceptable Trade-off of Relative Error (10%) for more efficient prediction on optimized parameters to obtain the optimized key rate

Conclusion of the Project

01 **Challenges and Solution**

- Optimization is too slow for real-time QKD
- Neural Network inferences provides ~270x speedup
- Implication: NN offers a practical efficient solution for real-time, adaptive tuning, especially on-low power platforms

02 **NN Accuracy and Validation**

- Accurately predicts optimized parameters & SKR across trained range and generalizes well to unseen data.
- Maintains high fidelity: SKR relative error remains low (<6%) even near transmission limits, validating reliable performance without significant loss of key rate.

03 **Future Directions**

- Refine NN accuracy for edge cases (near transmission limits)
- Incorporate more realistic device models for enhanced relevance

KTH VETENSKAP OCH KONST

Reference

[1] Charles Ci Wen Lim et al. "Concise security bounds for practical decoy-state quantum key distribution". In: Physical Review A 89.2 (2014), p. 022307.

[2] Charles H Bennett and Gilles Brassard. "Quantum cryptography: Public key distribution and coin tossing". In: Theoretical computer science 560 (2014), pp. 7–11.

[3] Artur K Ekert. "Quantum cryptography based on Bell's theorem". In: Physical review letters 67.6 (1991), p. 661.

[4] Y. Xiang et al. "Simulated Annealing with GPU Acceleration". In: IEEE Transactions on Parallel and Distributed Systems 24.8 (2013). Accessed: March 18, 2025, pp. 1568–1577.

[5] C. Tan et al. Optimization Algorithms for Real-Time Systems. Accessed: March 18, 2025. Amsterdam, Netherlands: Elsevier, 2018. isbn: 9780123456789.

[6] Weizhao Lu et al. "Recurrent neural network approach to quantum signal: coherent state restoration for continuous-variable quantum key distribution". In: Quantum Information Processing 17 (2018), pp. 1–14.

[7] Weiqi Liu et al. "Integrating machine learning to achieve an automatic parameter prediction for practical continuous-variable quantum key distribution". In: Physical Review A 97.2 (2018), p. 022316.

[8] Wenyuan Wang and Hoi-Kwong Lo. "Machine learning for optimal parameter prediction in quantum key distribution". In: Physical Review A 100.6 (2019), p. 062334.

[9] Hiking and Coding. "A Cascade Information Reconciliation Tutorial". In: Hiking and Coding (Jan. 2020). Accessed: 2025-04-15. url: https://hikingandcoding.com/2020/01/15/a- cascadeinformation-reconciliation-tutorial/.

[10] V. Scarani et al. "The Security of Practical Quantum Key Distribution". In: Reviews of Modern Physics 81 (2009), pp. 1301–1350.

[11] Chi-Hang Fred Fung, Xiongfeng Ma, and HF Chau. "Practical issues in quantum-key-distribution postprocessing". In: Physical Review A—Atomic, Molecular, and Optical Physics 81.1 (2010), p. 012318.

[12] Christoph Roser. Local Optima Global Optimum. Illustration. Available at: https://www.allaboutlean.com/ (accessed: March 04, 2025), Licensed under Creative Commons Attribution-ShareAlike 4.0 International License (CC-BY-SA 4.0), https://creativecommons.org/licenses/by-sa/4.0/. July 2018. url: https://www.allaboutlean.com/.

[13] John A Nelder and Roger Mead. "A simplex method for function minimization". In: The computer journal 7.4 (1965), pp. 308–313.

[14] Deep Learning Demystified. "Introduction to Neural Networks - Part 1". In: Medium (May 2023). Accessed: 2025-04-15. url: https://medium.com/deep-learning-demystified/introductionto-neural-networks-part-1-e13f132c6d7e.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Accessed: March 18, 2025. Cambridge, MA: MIT Press, 2016. isbn: 9780262035613. url: http://www.deeplearningbook.org/.

[16] Guangyu Yang and Xiao-Jing Wang. Artificial neural networks for neuroscientists: A primer. June 2020. doi: 10.48550/arXiv.2006.01001.

[17] Renyi Cai and Valerio Scarani. "Finite-Key Analysis for Practical Implementations of Quantum Key Distribution". In: New Journal of Physics 11.4 (2009). doi: 10.1088/1367- 2630/11/4/045024. url: https://doi.org/10.1088/1367-2630/11/4/045024.

[18] SciPy Developers. SciPy Documentation: dual annealing. Accessed: March 18, 2025. 2023. url: https : / / docs . scipy . org / doc / scipy / reference / generated / scipy . optimize . dual _ annealing.html.

[19] Lester Ingber. "Simulated Annealing: Practice versus Theory". In: Mathematical and Computer Modelling 18.11 (1993). Accessed: March 18, 2025, pp. 29–57.

[20] Apple Inc. Apple Silicon Overview. Accessed: March 18, 2025. 2022. url: https://www.apple.com/silicon-overview/.

[21] TensorFlow Team. TensorFlow Lite on ARM. Accessed: March 18, 2025. 2023. url: https://www.tensorflow.org/lite/guide.

[22] Apple Inc. Metal Programming Guide. Accessed: March 18, 2025. 2022. url: https://developer.apple.com/metal/.

Thank you for your time!

Presentation Slides

Project Document

Q&A

Further Questions?

Please reach out to

slleung@kth.se

+46 0723751735