Using Neural Networks to Predict G/G/S Queue Performance from Synthetic Datasets

Author: Aashrith Raj Tatipamula
Placement: King's University College
Department: Management, Economics and Mathematics (MEM)
Co-op Teacher: Mr. Laszlo
Supervisor: Dr. Felipe Rodrigues
Date: July 23rd 2025

2. Abstract

A G/G/s queue is a type of system where customers arrive randomly and are served for varying durations, and are attended to by multiple servers. Predicting performance metrics like Wq (average wait time in queue) and Lq (average number in queue) is very important for optimizing operations in high-traffic environments such as hospitals, banks, and call centers. This project generated 10,000 synthetic queuing scenarios using Excel VBA, simulating arrival and service rates. Machine learning models including neural networks built using TensorFlow/Keras were trained to predict Wq and Lq based on input parameters. The findings found in this project show that machine learning can replace traditional queueing formulas for faster, more scalable performance in complex, real-life systems.

3. Introduction

This project explores how machine learning can help us better understand and manage the queueing system behind many real-world services like hospitals, banks, and call centers. These systems model how people or tasks line up, wait, and get served. Traditionally, models like M/M/s queues assume everything is neat and mathematical: arrivals and service times follow exponential (predictable) patterns. But real life is messy. People don't show up or get helped at perfectly random intervals. That's where the G/G/s queue comes in. It allows general (any kind of) arrival and service time distributions, plus multiple servers, making it much more realistic. The goal of this project is to accurately predict Wq, the average waiting time in line. Instead of relying on complicated formulas and approximations, I trained three machine learning models on data generated through simulated G/G/s systems. This approach offers a fast, flexible way to predict queue performance, helping decision-makers in service industries save time, reduce congestion, and improve experiences.

4. Theoretical Background

A queue is just a fancy word for a line whether it's people waiting at Tim Hortons, patients at a hospital, or data packets on the internet. Queueing theory helps us understand how long people wait, how many are in line, and how busy the service area is. To do this, we look at some key input variables:

- λ (lambda) = average arrival rate → This is how many people or tasks arrive per time unit (e.g., 10 patients/hour).
- μ (mu) = average service rate → This is how many can be served per time unit (e.g., 5 patients/hour per doctor).
- s = number of servers →This is how many people or machines are serving the queue (e.g., 3 doctors).
- ρ (rho) = utilization →This is how busy the system is, calculated as $\rho = \lambda / (s \times \mu)$.

- Lq = Length of Queue → This is how many people are in the queue

To calculate the output of Wq (average wait time in queue) in the dataset, I used some approximation formulas designed for G/G/s systems.

However, analytical methods have limitations. They're based on steady-state assumptions, which means the system never changes, runs forever, and behaves perfectly which isn't how real life works. In hospitals, for example, patients may arrive in bursts, or service may vary depending on urgency. That's why machine learning is a better alternative. ML models like neural networks can learn from real or simulated data and make fast, accurate predictions even when the math is too complex or the assumptions don't hold.

5. Methodology

5.1 Data Generation

To build the predictive models, I created a synthetic dataset of 10,000 queueing scenarios using Excel VBA.

- **Input ranges:**
    - **ρ (utilization):** 0.5 – 0.99
    - **λ (arrival rate):** 1 – 20
    - **s (number of servers):** 1 – 10
    - **μ (service rate):** Derived using: $\mu = \lambda / (\rho \times s)$
    - **Lq:** Derived using $Lq = \lambda \times Wq$

- **Outputs:**
    - **Wq:** Estimated using the **Allen-Cunneen** approximation

$$W_q = \frac{1}{s\mu} \cdot \frac{\rho\left(\sqrt{2(s+1)} - 1\right)}{1 - \rho}$$

To generate the output variable Wq (average wait time). I used analytical approximation formulas based on queuing theory for G/G/s systems (e.g., Allen-Cunneen approximation), since full-scale simulations for each point would be too time-consuming.

5.2 Feature Engineering & Preprocessing

- **Input Features:** λ, μ, s, ρ, Lq
- **Output Variable:** Wq
- **Normalization:** StandardScaler applied
- **Data Split:** 70% training, 30% testing

5.3 Machine Learning Setup

Architecture:

- Neural Network (NN): 3 hidden layers with 32, 16, and 1neurons
- Activation functions: ReLU for hidden layers, Softplus for output

Loss Function:

- Mean Squared Error (MSE)

Optimizer:

- Adam Optimizer with learning rate 0.001

Framework Used:

- Keras with TensorFlow backend for NN
- Scikit-learn for Random Forest
- XGBoost for Gradient Boosted Trees

Training Process:

- Epochs: 100 for NN
- Batch Size: 32
- Hardware/Software: Python, Intel i7 CPU, 16GB RAM, Windows 10

6. Results

6.1 Performance Metrics (Test Set)

Prediction Performance (Testing):

- **Neural Network**
  - MAE: 0.0959
  - MSE: 0.0729
  - RMSE: 0.2700
- **Random Forest**
  - MAE: 0.0342
  - MSE: 0.0642
  - RMSE: 0.2533
- **XGBoost**
  - MAE: 0.0342
  - MSE: 0.0642
  - RMSE: 0.2533

7.2 Visualizations

- **Scatter Plots:** Predicted vs Actual Wq values closely aligned, especially for RF and XGBoost.
- **Residual Analysis:** Neural network showed greater variance; tree-based models had tighter error distributions.
- **Learning Curve:** Neural network's validation loss stabilized after ~60 epochs with mild signs of overfitting.

8. Discussion

8.1 Interpretation

- The neural network achieved high correlation but required careful tuning and was more sensitive to noise.
- Random Forest and XGBoost outperformed the neural net in accuracy and stability with minimal tuning.
- The models effectively captured non-linear relationships among variables.

8.2 Limitations

- Used **synthetic data** only, no real-world data.
- Assumed **steady-state behavior** (no dynamic or time-varying queues).
- Focused on G/G/s queues only excluded finite buffers (G/G/s/K), priorities, or blocking.

8.3 Potential Improvements

- Try ensemble stacking or attention models (Transformers)
- Apply models to real-world healthcare/hospital queuing data
- Add support for time-dependent features as dataset relied on steady state queues

9. Conclusion

9.1 Summary

This project set out to predict queuing metrics like Wq using machine learning trained on simulated G/G/s queue data. The results show that:

- Tree-based models (RF, XGBoost) outperformed neural networks
- ML models can serve as fast, reliable predictors even for complex queueing behavior
- Neural networks are promising but need hyperparameter tuning and larger datasets

9.2 Key Takeaways

- ML is a viable alternative to traditional queue theory approximations
- Synthetic data can still produce accurate models
- Model performance varies significantly by algorithm choice
- Neural networks are flexible but not always best without optimization
- Random Forest and XGBoost offer strong baselines for tabular regression problems

9.3 What I Learned

- Built a complete ML pipeline from **data generation → prediction → evaluation**
- Deepened understanding of **queuing systems** and their real-world significance
- Gained hands-on experience in **modeling**, **feature engineering**, and **Simul8**

Future Work:

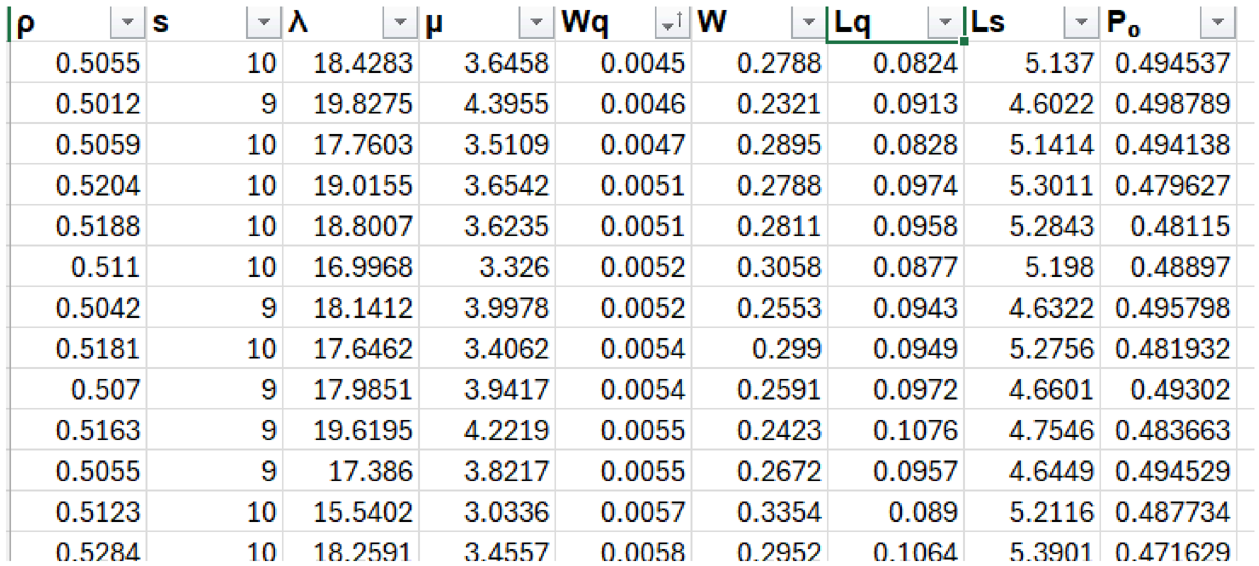
- Extend to G/G/s/K queues with finite buffers
- Add arrival/service distributions that change over time
- Build and deploy a real-time Wq prediction web app/tool
- Make DES in Simul8 actually work

10. References

- **Rodrigues, F. F.**; Kashef, R.: A note on queuing models. Ivey publishing reference 9B19E006. May 2019. https://www.iveycases.com/ProductView.aspx?id=102463
- Scikit-learn and Keras official docs
- XGBoost Python API

11. Appendices

A. Sample Dataset (first 5 rows)



ρ	s	λ	μ	Wq	W	Lq	Ls	P_o
0.5055	10	18.4283	3.6458	0.0045	0.2788	0.0824	5.137	0.494537
0.5012	9	19.8275	4.3955	0.0046	0.2321	0.0913	4.6022	0.498789
0.5059	10	17.7603	3.5109	0.0047	0.2895	0.0828	5.1414	0.494138
0.5204	10	19.0155	3.6542	0.0051	0.2788	0.0974	5.3011	0.479627
0.5188	10	18.8007	3.6235	0.0051	0.2811	0.0958	5.2843	0.48115
0.511	10	16.9968	3.326	0.0052	0.3058	0.0877	5.198	0.48897
0.5042	9	18.1412	3.9978	0.0052	0.2553	0.0943	4.6322	0.495798
0.5181	10	17.6462	3.4062	0.0054	0.299	0.0949	5.2756	0.481932
0.507	9	17.9851	3.9417	0.0054	0.2591	0.0972	4.6601	0.49302
0.5163	9	19.6195	4.2219	0.0055	0.2423	0.1076	4.7546	0.483663
0.5055	9	17.386	3.8217	0.0055	0.2672	0.0957	4.6449	0.494529
0.5123	10	15.5402	3.0336	0.0057	0.3354	0.089	5.2116	0.487734
0.5284	10	18.2591	3.4557	0.0058	0.2952	0.1064	5.3901	0.471629

B. Code Repository

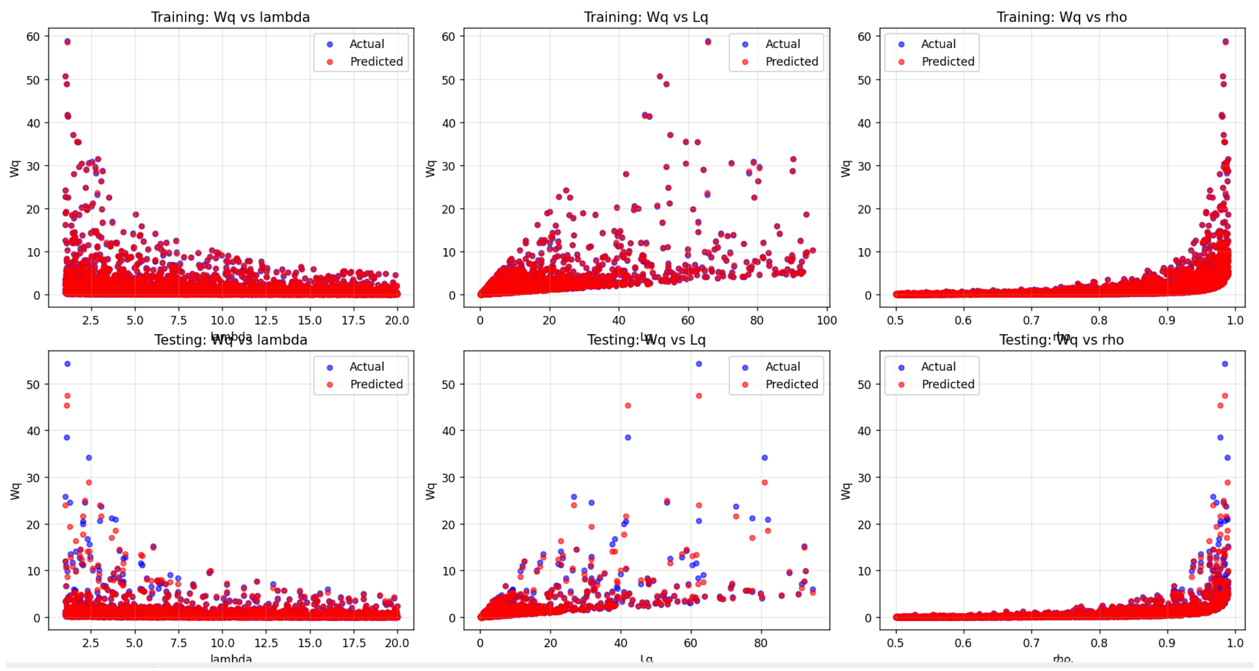
GitHub: https://github.com/Tabulater/KingsUnivieristyCollege

C. Approximation Formulas

- **Allen-Cunneen for Wq**

- **Lq = λ × Wq**

D. Chart/Plots


Training: Wq vs lambda
Training: Wq vs Lq
Training: Wq vs rho
Testing: Wq vs lambda
Testing: Wq vs Lq
Testing: Wq vs rho
Actual
Predicted
Wq
lambda
Lq
rho