mathematics

MDPI

Article

Use Cases of Machine Learning in Queueing Theory Based on a $GI/G/K$ System

Dmitry Efrosinin [1,*], Vladimir Vishnevsky [2], Natalia Stepanova [1] and Janos Sztrik [3]

[1] Institute for Stochastics, Johannes Kepler University Linz, 4040 Linz, Austria; natalia0410@rambler.ru
[2] V. A. Trapeznikov Institute of Control Sciences, Moscow 117997, Russia; vishn@inbox.ru
[3] Department of Informatics and Networks, Faculty of Informatics, University of Debrecen, 4032 Debrecen, Hungary; sztrik.janos@inf.unideb.hu
* Correspondence: dmitry.efrosinin@jku.at

Abstract: Machine learning (ML) in queueing theory combines the predictive and optimization capabilities of ML with the analytical frameworks of queueing models to improve performance in systems such as telecommunications, manufacturing, and service industries. In this paper we give an overview of how ML is applied in queueing theory, highlighting its use cases, benefits, and challenges. We consider a classical $GI/G/K$-type queueing system, which is at the same time rather complex for obtaining analytical results, consisting of K homogeneous servers with an arbitrary distribution of time between incoming customers and equally distributed service times, also with an arbitrary distribution. Different simulation techniques are used to obtain the training and test samples needed to apply the supervised ML algorithms to problems of regression and classification, and some results of the approximation analysis of such a system will be needed to verify the results. ML algorithms are used also to solve both parametric and dynamic optimization problems. The latter is achieved by means of a reinforcement learning approach. It is shown that the application of ML in queueing theory is a promising technique to handle the complexity and stochastic nature of such systems.

Keywords: $GI/G/K$ queueing system; performance analysis and optimization; machine learning; dynamic programming; reinforcement learning; deep Q-learning

MSC: 60K25; 60K30; 90C40; 68T20

Academic Editor: Ivan Atencia and José Luis Galán-García

Received: 4 January 2025
Revised: 4 February 2025
Accepted: 25 February 2025
Published: 26 February 2025

Citation: Efrosinin, D.; Vishnevsky, V.; Stepanova, N.; Sztrik, J. Use Cases of Machine Learning in Queueing Theory Based on a $GI/G/K$ System. *Mathematics* **2025**, *13*, 776. https://doi.org/10.3390/math13050776

1. Introduction

Machine learning (ML) offers powerful tools to address challenges in queueing theory, particularly in optimizing performance, predicting behavior, and improving decision-making in systems involving queues. ML can be defined as a set of techniques that automatically identify patterns in data and then use them to predict future data or make decisions under uncertainty. ML can be broadly classified into three types: supervised, unsupervised, and reinforcement learning [1]. In supervised ML methods [2], the goal is to learn the mapping from input (features) to output (values of the target function), given a set of observable input–output pairs. The two types of problems that supervised ML is designed to solve are classification problems and regression problems. Unsupervised learning (see, e.g., [3]) is a type of ML where the model learns patterns and structures in data without any labeled outputs or specific guidance. Unlike supervised learning, this type of ML works solely with input data to uncover hidden patterns, relationships, or structures. Common tasks of this type of ML include clustering, dimensionality reduction,

anomaly detection, and density estimation. Reinforcement learning (RL) (see, e.g., [4]) is used to solve dynamic optimization problems, i.e., to learn what to do and how to influence the system by periodically receiving rewards or incurring costs. Reinforcement learning is concerned with finding appropriate control actions in a particular situation in order to maximize rewards or minimize costs. In this case, the learning algorithm is not given any examples of optimal outputs, unlike the supervised ML approach. Instead, it must find the optimal solution by trial and error, gradually modifying the appropriate set of control actions. A recent trend is to combine reinforcement learning with deep learning algorithms such as neural networks. This hybrid method is called deep reinforcement learning. All of the above types of machine learning can be applied in queueing theory. However, the use of supervised learning algorithms and reinforcement learning will be the dominant topic in our paper.

The main motivation for writing this article was the numerous discussions of the authors with experts in classical methods of queueing theory. We noticed that some of them do not quite understand in what context machine learning methods can be applied in queueing problems, and they treat these methods with a certain skepticism. Unfortunately, we have not been able to find any paper that summarizes the relevant tasks and methods combining the machine learning approach with performance analysis and optimization of queueing systems. The next section will provide an overview of the available related publications, but these works focus mainly on solving a specific problem. As for optimization problems, e.g., dynamic optimization by reinforcement learning, there are practically no such works on queueing systems. Thus, the main idea of the paper is to bring together and solve special problems in a simple and compact form for which machine learning can be effectively applied to solve classification and regression problems and to optimize such systems statically and dynamically. The results presented here will help experts in the field of queueing theory get acquainted with new possibilities and broaden their horizon of knowledge. The training and test data sets were generated using a simulation model, followed by their application for standard ML algorithms. We want to answer the question of whether the noisy data generated by the simulation could be suitable to obtain the high-quality trained ML algorithms. We do not set here the task of developing new machine learning algorithms, finding their optimal architecture, improving the quality of features in data sets, building new approximation models for time distributions, etc. Here, we have collected the main directions where machine learning can be applied in queueing theory, in principle, for almost any arbitrary queueing system and give some recommendations.

As a basic queueing model in this paper, we consider a queueing system of a general classical type $GI/G/K$ with one common queue, K homogeneous servers, independent equally distributed times between customer arrivals, and independent equally distributed service times with arbitrary distributions. Such a system has many potential applications in computer, telecommunication, and manufacturing systems. This queueing system is also well suited for demonstrating the effectiveness of combining the simulation required to generate training and test samples with ML algorithms to provide performance analysis and optimization. For this system, it is generally difficult to obtain analytical results, but there are nevertheless relationships derived from approximation analysis of the system that can be applied to verify the estimates obtained. This approach can be used to investigate arbitrary complex queueing models. However, the success of ML implementation depends on robust data, model validation, and careful integration into operational systems. It should be noted that the initial accumulation of information when creating the necessary samples, of course, requires considerable time expenditure. However, once trained, ML models provide instantaneous predictions of the characteristics studied for new values of the input parameters of the system without the need for a new statistical data simulation. Although

it is not the purpose of this paper to describe existing methods for approximating the performance of the system we are considering, we will nevertheless give a brief overview of the available results that can be used to verify the computation of system performance estimates via ML. There are several approaches to obtaining approximate results for a system of type $GI/G/K$, e.g., approximating the service time distribution by a phase-type distribution (PH distribution) in the $M/G/K$ system, approximating the continuous-time model by an equivalent discrete-time model, and using a two-moment approximation based on the exact solution for $GI/D/K$ and $GI/M/K$ systems. More details on these methods can be found, for example, in papers such as [5–10]. In [11], the approximation of a more complex $GI/G/K$ system with retrial customers was investigated using a system of $PH/PH/K$ where the parameters of the PH distribution were estimated from the first three moments. In this work, the authors numerically investigated the effect of the time between customer arrivals and service on the performance characteristics of the system. It was found that the functions describing the performance of the queueing system are more sensitive to changes in the moments of interarrival times than to the moments related to the service time.

In this paper, we approximate the $GI/G/K$ queueing system with the equivalent $PH/PH/K$ queue. The computation of the performance characteristics of this system with PH time distributions will be performed through simulation. Since most of the obtained results for approximating the average performance characteristics depend on the first two moments, we will also use features that depend on the first two moments in the training sample. More specifically, we plan to use the feature vector in training and test samples, which includes the system load factor as the ratio of the average service time to the average interval between arrivals, as well as the squares of the coefficients of variation of the time between arrivals of customers and their service. In place of the target value, the system performance characteristic in which we are interested will be used. The basic idea is to use the data obtained by simulation to train ML algorithms, which could then be used to perform analyses of $GI/G/K$ systems with given arbitrary specified distributions or corresponding moments. The task of training sample preparation is a separate topic. In principle, it would be possible to vary the values of the moments in a step and then estimate the parameters of the PH distributions from their values (see, for example, [12]). However, this approach requires at least three moments for a PH distribution with two transition states and five moments for three transition states. Moreover, it is not possible to estimate the parameters of the PH distribution for every set of moments. Thus, we propose to generate training samples by randomly selecting from a given interval the values of the parameters of the PH distributions, calculate the moments for them, and put these moments in correspondence with the values of the performance characteristics of the studied queueing system. The approach presented here is quite universal and can be used for queueing systems with arbitrary architectures and distributions of times between events occurring in the system. This paper shows typical examples of applying supervised machine learning to solve regression and classification problems arising in system performance and reliability analysis, as well as parametric optimization. We also demonstrate an example of applying reinforcement learning to a dynamic optimization problem.

The remainder of the paper is organized as follows. Section 2 covers the related publications. In Section 3 we represent the main queueing model and data generation. Section 4 is devoted to the simulation algorithms. The typical regression and classification tasks are enumerated and discussed, respectively, in Sections 5 and 6. The reinforcement learning approach is demonstrated in Section 7. The final conclusions are given in Section 8.

2. Related Publications

Although the first steps in the successful application of machine learning for performance evaluation of simple and complex queueing systems have already been made, the total number of papers on this topic in the world literature remains modest. In terms of reviews, we can only refer to recent works [13] that offer a small systematic introduction to the use of machine learning in the study of queueing systems and networks. The paper [14] investigated the possibility of using time series data that describe the behavior of the $M/M/s$ queue to obtain, using artificial neural networks, a prediction of various characteristics, such as the number of customers in the system at a given time interval. In [15,16], the authors ask whether machine learning, namely neural networks, can be fundamentally useful for analyzing systems of type $GI/G/s$ and $M/G/1$, respectively. In the first case, the authors trained a neural network on the first two moments of time between customer arrivals and service to estimate characteristics such as average waiting time and blocking probability for a finite buffer system. In the second case, a simpler system was considered, but a global problem was solved, namely the estimation of the stationary distribution of queue length over the arrival rate and first five moments of the service time. Positive results were obtained, indicating the potential of using the proposed approach for more general systems. In [17], a two-layer neural network was used to predict the waiting time in the queueing system. Markov queues were modeled using artificial neural networks in the work [18]. A study by [19] focused on predicting the waiting time of a patient in an emergency department using a deep learning algorithm based on real data on the severity of the patient's condition collected from electronic medical records. Various ML models and methods used in queueing theory have been discussed in [20], applied to the multichannel $M/G/n$ queueing system. In addition, in [21], a neural network model was developed to efficiently approximate the performance of various multichannel tandem queueing systems with a finite buffer, where the individual tandem nodes differed both in service rates and in the number of servers. In [22,23] an approach based on a combination of machine learning and simulation methods was first applied to studying the multi-priority $MMAP/PH/M/N$ queueing system. A method for estimating the response of the system, probability of loss of customers, and other metrics of a large-dimensional tandem queueing system with a correlated input flow, finite-capacity buffer and using machine learning algorithms was described in [24–26]. The effective application of ML methods has been demonstrated by [27,28], who analyzed complex stochastic polling systems with different disciplines of adaptive and cyclic polling. In [29], ML was used to estimate the maximum queue length for the queueing system during a busy period based on information about the number of tagged customers. In [30], the authors used machine learning and data obtained by simulation to calculate stationary performance measures of a queueing system with splitting and merging of customers (a fork-join system with the MAP arrival flow). A finite source queueing system is considered in [31] to analyze a k-from-n reliability system that adequately models the operation of unmanned multirotor drones. Here, as before, machine learning is applied to predict the reliability characteristics of this system. In [32], the authors presented a machine learning tool to predict the mean number of customers in the $M/M/c/K$ queueing system with a finite buffer. The peculiarity of this work lies in its development of supervised learning based on the Michaelis–Menten nonlinear model used in biochemistry.

We managed to find some papers in the field of applying ML to optimal control problems in queueing theory. The problem of optimal scheduling in a single-channel system with parallel queues and with time intervals for switching a server to another queue is described in [33]. In this system, a learning vector quantization (LVQ) network was used for scheduling control; see [34] for details. The combination of a Markov decision problem

and a neural network for a heterogeneous queueing model with processor sharing was studied in [35]. The effective use of a neural network for the selection of control actions in the problem of optimal scheduling of a single server in a general system with several parallel queues was demonstrated in [36]. Gradually, works in which machine learning methods found their application in the problems of dynamic control in queueing systems began to appear. Until recently, the main methodology of dynamic optimization and the study of the structural properties of optimal control in queueing theory were dynamic programming algorithms based on the Bellman principle. Examples of the use of these algorithms for computing and analyzing control policies of queueing systems described by Markov (MDP) and semi-Markov (SMDP) decision processes can be found in numerous publications, including those of the authors of this paper; see, for example, [37,38]. In recent years, to solve dynamic optimization problems, reinforcement learning methods have been actively developed and implemented. In RL, an agent learns to make decisions by taking actions in an environment to maximize the average reward or minimize the average cost. The agent interacts with the environment in a trial-and-error manner, learning from the feedback (rewards or costs) received for its actions. The environment of queueing systems is the state space where the states normally represent the status of the queue and servers, actions represent decisions (e.g., scheduling of servers for multiple parallel queues, routing of customers between parallel queueing systems, resource allocation in the form of heterogeneous servers), and rewards or costs represent performance metrics. The RL technique includes a large number of diverse algorithms, such as Q-learning (QL) [39], deep Q-learning (DQN) [40], double DQN (DDQN) [41], policy gradient method (PG) [42], actor-critic methods (AC) [43], and deep deterministic policy gradients (DDPG) [44], among others. All of these algorithms belong to the so-called off-policy model-independent class of RL algorithms. This means that the agent can learn from historical data generated by any policy, not just the one it is currently following. In [45], the DQN method was implemented for admission control in a 5G wireless network. The AC algorithm for optimal control in a heterogeneous queueing system was discussed in [46]. Comparison analysis of RL algorithms and random search methods for the finite source queueing system with heterogeneous servers in the repair facility was proposed in [47]. Model-dependent RL algorithms, where the structure of controlled queueing systems was taken into account, were studied in [48].

3. Main Model and Data Generation

Consider the $GI/G/K$ queueing system, which is a classical general and versatile model used in queueing theory to analyze systems where customers or tasks arrive, wait in a queue if necessary, and are served by one of the K servers. The time between arrivals (or interarrival times) follows a general distribution, denoted as GI. That is, these interarrival times are independent and identically distributed. The distribution of the interarrival time is characterized by the first two moments defined by the arrival intensity λ, where λ^{-1} specifies the mean interarrival time and CV_a^2 is the squared coefficient of variation. This characteristic is used instead of the second moment or variance because it is dimensionless and can be interpreted independently of the first moment. The service time for each customer is drawn from a general distribution, also denoted as G. Similarly, it is assumed that the distribution of service times is characterized by the mean b and the corresponding square of the coefficient of variation CV_s^2. Thus, to simplify the data structure, we will describe the parameters of the queueing system by means of a three-dimensional tuple (ρ, CV_a^2, CV_s^2) for each fixed value of the number of servers K, where $\rho = \frac{\lambda b}{K}$ represents the load factor of the system. It is assumed that our system is stable, meaning the condition for the existence of an ergodic regime is fulfilled; that is, $\rho < 1$.

Since the goal is to obtain a trained ML algorithm that would function stably for a number of distributions of interarrival and service times, it would be possible to generate values of random variables belonging to different parametric families of distributions with positive domains. We decided to simplify the problem and approximate the corresponding distributions by means of the fitted phase-type (PH) distributions. It is well known that any non-negative distribution can be arbitrarily well approximated by a PH distribution; relevant results can be found in [49–51]. An expectation maximization algorithm is often used to fit PH distributions to the data. The continuous-time PH distribution describes the random time T to absorption in a continuous-time Markov chain with l transition states and one absorbing state. The PH distribution is characterized by two hyper parameters: the vector of initial probabilities $\alpha = (\alpha_1, \ldots, \alpha_l)$ of size l to start in one of a transient state and the $l \times l$ subgenerator matrix $A = [a_{ij}]$ of transitions of the absorbing Markov chain defined in the state space of the process except for the absorbing ones. The distribution function PH is given by the relation

$$F_T(t) = 1 - \alpha e^{At} e, \tag{1}$$

where e is a all-ones column-vector with a size l. The corresponding moments are defined as

$$\mathbb{E}[T^n] = (-1)^n n! \alpha A^{-n} e. \tag{2}$$

In practice, however, the approximation may be inaccurate if the size of the approximating process is fixed. There are some challenges with respect to the approximation process. The number of states l in the PH distribution affects the quality of the approximation and the computational complexity. Capturing heavy-tailed distributions or specific skewness may require many states, but optimization during fitting can become numerically unstable for large l. In some cases, it is better to approximate some arbitrary distribution by a mixture of phase-type distributions (1),

$$F_T(t) = \sum_{j=1}^{J} \beta_j F_{T_j}(t) = 1 - \sum_{j=1}^{J} \beta_j \alpha_j e^{A_j t} e \text{ with } \sum_{j=1}^{J} \beta_j = 1,$$

where J denotes the maximum order of the PH distribution. The machine learning algorithms presented in this work are trained in parametric distribution families. It is clear that there are empirical data for which the PH distributions with fixed dimension are not suitable, especially if we are talking about distributions with heavy tails, which, however, does not happen very often in queueing theory. In this case, it makes sense to use PH distributions with higher dimensions taking into account that it can be computationally expensive and can overfit the data, leading to poor generalization, or use other parametric models with fewer parameters. In our model, it is also assumed that the times between request arrivals and service times are independent random variables. However, the simulation models used are not limited to these models. In principle, a similar approach can be used for correlated arrival flows, such as the Markov arrival process, and in addition to the first two moments, the correlation coefficient with lag 1 can be specified as an element of the feature vector.

Let us define by $x = (x_1, \ldots, x_n)$ the feature vector and by y the value of the target function. Then, the set D of generated observable pairs or examples can be represented as

$$D = D_L \cup D_T = \{(x^{(j)} \to y^{(j)}) : j = 1, \ldots, |D|\},$$

where $|\cdot|$ applied to a finite set denotes the number of its elements. Here, D is the set of examples divided into a training subset D_L and a test subset D_T, and $|D| = |D_L| + |D_T|$ is

the total number of pairs summed from the number of examples in the training and test subsets. Simulation-based data set generation is a possible method for training and testing machine learning models, particularly in queueing systems where the use of analytical results, experimentation, and data collection is infeasible. As shown in the next section, by tailoring the simulation design to the problem at hand, it is possible to create meaningful and high-quality data sets for various concrete tasks. Algorithms of simulation models as well as machine learning algorithms were implemented in the computational software system *Mathematica* 14.0 © of Wolfram Research (Champaign, IL, USA).

4. Simulation Techniques

Simulation techniques are methods used to model real-world processes or systems in a virtual environment to analyze their behavior under various scenarios. For our purposes, we use two types of simulation methods: event-based simulation and simulation by moments of departure. The event-based simulation is a powerful tool that can be used to estimate different system-level metrics such as stationary state distribution, average number of customers in the system and in the queue, average length of the busy period, the blocking probability, the probability of an idle system and the customer-level characteristics (for example, average waiting and sojourn times), and the corresponding moments which can be used to approximate the waiting time and sojourn time distributions. Simulation by moments of departure can also be used to calculate the stationary state distribution and to estimate the average characteristics. This algorithm is much faster than the previous one but is not suitable for estimating the higher moments of the waiting time required to approximate the distribution function of the stationary waiting and sojourn times.

4.1. Event-Based Simulation

Discrete event simulation is a technique suitable for studying processes in which state changes occur at random moments in time rather than continuously. In this case, it is sufficient to observe the process only at the moments of time when the state changes occur, which are called events. For a $GI/G/K$-type queueing system, the events are the moments when new customers arrive to the system and the service completion moments, if the system is not empty. In this case, there is a change in the number of customers in the queue, which we denote by n. The change in this state is completely determined by the residual times until the next arrival and until the end of service on the servers if they are busy. This is very similar to the method of introducing additional variables by analyzing non-Markovian processes. This approach is quite universal and can be used to build a simulation model of the queueing system with arbitrary distributions of random variables that describe the dynamics of the system.

Denote by x_a the residual time until the next customer arrives and by $x_b = \{x_{b,k} : k = 1, 2, \ldots, K\}$ the residual times until the service is completed on each of the busy servers. In cases where server j is idle, we assume $x_{b,j} = \{\infty\}$. Note that it is not necessary to store the states of the idle servers, since it is assumed that they are identical and therefore the service time does not depend on the index of the server used. Thus, the vector x_b contains only the residual service times of the busy servers. The change in the number of customers in the system is entirely determined by the vector state $x = (n, x_a, x_b)$. To initialize the simulation model, we use the initial state $x_0 = (0, x_a, x_b) = (0, 0, \{\infty\})$ when there are no customers in the system and no residual service time. In the approach proposed here for building the simulation model, all time variables will be relative, i.e., defined relative to the event at the present time. We set the total simulation time $T_s = 0$ and the time-weighted number of customers $T_n = 0$. We need these variables to estimate the average performance of the system. If the vector state becomes $x = (n, x_a, x_b)$ after the next state

change in the number of customers, then the dwell time in the current state T_x is defined as $T_x = \min\{x_a, x_b\}$. The components of the residual times to the next event are then redefined as follows,

$$x_a \leftarrow x_a - T_x,\ x_b \leftarrow x_b - T_x,\ \text{ for } x_{b,k} < \infty,\ k = 2, \ldots, |x_b| \leq K + 1.$$

If after this operation $x_a = 0$, then the next event is related to the arrival of a new customer, i.e., $n \leftarrow n + 1$. In this case, the component x_a is replaced by the generated new value of the time to the next arrival T_a, $x_a \leftarrow T_a$. Moreover, if there are idle servers in the system after the arrival of a new customer, $n < K$, then a new service time T_b is also generated and added to the vector x_b, i.e., $x_b \leftarrow x_b \cup T_b$. In this case, the new state of the system becomes the state

$$y = (n + 1, x_a, x_b).$$

If $x_{b,j} = 0$ for some server $j = \arg\min\{x_b\}$, then the next event corresponds to the end of service on this server, i.e., $n \leftarrow n - 1$. In this case, component j of vector x_b is removed, $x_b \leftarrow x_b \setminus \{x_{b,j}\}$. If there are any pending customers, $n \geq K$, left in the queue after the end of service, a new service time T_b is generated, which is again added to the vector x_b, $x_b \leftarrow x_b \cup T_b$. Otherwise, vector x_b remains the same. The state of the system at the end of service becomes

$$y = (n - 1, x_a, x_b).$$

Next, the dwell time T_y in state y is calculated, $T_y = \min\{x_a, x_b\}$, and further redefining the values of x_a and x_b, the process continues further. The simulation algorithm can be terminated either by the total system observation time T_{max} or by the number N_{max} of customers arrived to the system. To ensure the independence of the process trajectory from the initial state, we record the required values $T_n \leftarrow T_n + nT_x$ and $T \leftarrow T + T_x$ after at least N_{min} customers have entered the system. The sequence of steps of the proposed method is described as the flowchart diagram in Figure 1 and as the pseudo-code in the Algorithm 1. Using this algorithm and the entries for variables T_n and T, we can calculate the average number of customers in the system,

$$\mathbb{E}[N] = \frac{T_n}{T_s}.$$

Little's formula also allows us to calculate the average sojourn time of a customer in the system, $\mathbb{E}[T] = \mathbb{E}[N]/\lambda$. Since $T = W + B$, where W is the waiting time of the customer in the queue and B is the service time, then $\mathbb{E}[W] = \mathbb{E}[T] - b$, and for the average number of customers in the queue, we obtain $\mathbb{E}[Q] = \lambda\mathbb{E}[W]$.

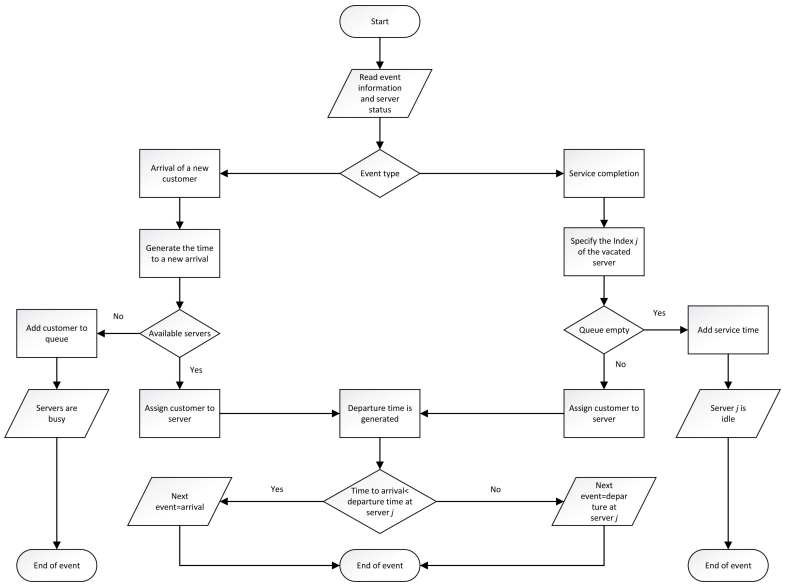

Figure 1. The flowchart of the event-based simulation algorithm.

Algorithm 1 Event-based simulation

Require: $x_a, x_b, F_A(t),\ F_B(t), N_{min}, N_{max}, T_{max},\ K$

$n = 0, x_a = 0, x_b = \{\infty\}, N_a = 0, T_s = 0, T_n = 0, T_x = 0$
while $N_a < N_{max} || T_s < T_{max}$ **do**
  $T_x = \min\{x_a, x_b\}$
  $x_a \leftarrow x_a - T_x, x_b \leftarrow x_b - T_x, x_{b,k} < \infty, k = 2, \ldots, |x_b|$
  if $N \geq N_{min}$ **then** ▷ Start recording values
    $T_n \leftarrow T_n + nT_x, T_s \leftarrow T_s + T_x$
  end if
  if $x_a \leq \varepsilon$ **then** ▷ Arrival of a new customer
    $N_a \leftarrow N_a + 1$
    $n \leftarrow n + 1$
    $x_a \leftarrow \text{Random}(F_A(t))$ ▷ The time to a new arrival
    if $n < K$ **then** ▷ If there are servers available
      $x_b \leftarrow x_b \cup \{\text{Random}(F_B(t))\}$ ▷ Add service time
    end if
  else if $\min\{x_b\} < \varepsilon$ **then** ▷ Service completion
    $j = \arg\min\{x_b\}$ ▷ Index of the vacated server
    $x_b \leftarrow x_b \setminus \{x_{b,j}\}$ ▷ Deleting an item from the list
    $n \leftarrow n - 1$
    if $n \geq K$ **then** ▷ If there are any waiting customers
      $x_b \leftarrow x_b \cup \{\text{Random}(F_B(t))\}$ ▷ Add service time
    end if
  end if
end while

4.2. Simulation by Moments of Departure

Here, we propose to use an Algorithm 2 based on the calculation of the departure time of a serviced customer as a simulation model of the queueing system. In this case, we denote the time of arrival of the ith customer to the system by $T_{a,i}$, and the time of its service by $T_{b,i}$. The total number of N_c customers that are generated at once. Such an algorithm is very simple and computationally efficient. According to this algorithm, the ith customer observes a set of times $x_{b,i} = \{x_{b,i,k} : k = 1, 2, \ldots, K\}$ representing the times when each of the servers will be available for service; that is, as in the previous algorithm, this vector describes the residual service times. That is, $x_{b,i}$ can be viewed as the state of the system when the ith customer arrives. Thus, the ith customer selects the server with index j_i from the set $x_{b,i}$, which will be idle and available for service earlier than the others, i.e., $j_i = \arg\min\{x_{b,i}\}$. The time for the ith customer to leave the system is then defined as

$$T_{d,i} = \max\{T_{a,i}, x_{b,j_i}\} + T_{b,i},$$

since either the server must wait for a customer to arrive or the customer must wait for the server to be released. The algorithm makes a preliminary sorting of the arrival times. Instead of assigning to each customer i a value $x_{b,i}$ from the $N_c \times K$ matrix x_b, the algorithm treats x_b as a continuously updated vector of length K representing the state of the system. In each iteration of the algorithm, we need to calculate the minimum element of the K dimensional vector $x_b \in \mathbb{R}_+^K$, which is used to determine the index of the server that must serve the ith customer, in code line 4. In terms of event-based simulation, the vector x_b represents the state of the system and the vector $T_a \in \mathbb{R}_+^{N_c}$ is a list of events, but these two objects fully describe all events in the system. Obviously, this feature distinguishes this algorithm from the previous classical event-based simulation algorithm. Usually, the queue length is the state of the system and the vectors T_a and T_d constitute a list of events, where the moments of arriving customers T_a are defined but the moments associated with the completion of the service must be constantly updated. Algorithm 2 can be used for simulation of a $GI/G/K$-type queueing system with an arbitrary number of servers. The interarrival and service times can be arbitrarily distributed, also without excluding a possible dependence between them.

Algorithm 2 Simulation by moments of departure

Require: $T_a \in \mathbb{R}_+^{N_c}, T_b \in \mathbb{R}_+^{N_c}, K \in \mathbb{N}, N_c$

1: Define the vectors $j \in \mathbb{N}^{N_c}, x_b \in \mathbb{R}_+^K$ and $T_d \in \mathbb{R}_+^C$
2: $i \leftarrow 1, x_{b,k} \leftarrow 0, k = 1, 2, \ldots, K$
3: **while** $i \leq N_c$ **do**
4: $\quad j_i \leftarrow \arg\min\{x_b\}$
5: $\quad x_{b,j_i} \leftarrow \max\{T_{a,i}, x_{b,j_i}\} + T_{b,i}$
6: $\quad T_{d,i} \leftarrow x_{b,j_i}$
7: **end while**
8: Arrange (T_a, T_d) in ascending order

As a result, the algorithm produces an ordered sequence of arrival times and service completion times. From these data, one can easily determine the number of customers in the system at an arbitrary point in time and, as a consequence, the values of such average system characteristics as the average number of customers in the system and in the queue, the average waiting time, the average number of occupied servers, as well as estimate the

probabilities of system states. However, unlike discrete event simulation, in this algorithm, there is no possibility to track the actual waiting times for each particular customer. The algorithm cannot estimate the values of the second- and higher-order moments of the waiting time of a customer in the queue and its sojourn time in the system.

4.3. Validation of the Simulation Model

In this section, we would like to verify the proposed simulation algorithms by comparing the calculations of the average number of customers in the system $\mathbb{E}[N]$ for different types of distributions with one of the approximation results. As the latter, we will take the simplest relation proposed in the monograph [6], namely

$$\mathbb{E}[N] = K\rho + \frac{\rho\sqrt{2(K+1)}}{1-\rho}\frac{CV_a^2 + CV_s^2}{2}. \tag{3}$$

Next, we turn to comparative analysis. Consider the $GI/G/10$ queueing system in which we will use the PH, gamma, Pareto, and lognormal distributions as distributions of the interarrival and service times. The parameters of these distributions are generated so that the mean values and variances coincide. The procedure for calculating the values for the system parameters and the subsequent comparative analysis includes the following main steps.

1. Initialize the system parameters $PH/PH/10$, where the PH distributions for the arrival stream of customers and for their service times are given by two transient states and are represented as

$$\vec{\alpha} = (\alpha, 1-\alpha),\ A = \begin{pmatrix} -a_{12}-a_{13} & a_{12} \\ a_{21} & -a_{21}-a_{23} \end{pmatrix},$$
$$\vec{\beta} = (\beta, 1-\beta),\ B = \begin{pmatrix} -b_{12}-b_{13} & b_{12} \\ b_{21} & -b_{21}-b_{23} \end{pmatrix},$$

   The probabilities α, β are chosen randomly from the interval $[0,1]$, and the intensities a_{ij} and b_{ij} from the interval $[0,2]$. A data set of parameter values $(\vec{\alpha}, A)$ and $(\vec{\beta}, B)$ with the length 125 satisfying the condition $\rho < 0.95$ is generated. Here, we deliberately exclude cases of heavy traffic, when the data obtained with the simulator have the highest variance.
2. We calculate the mathematical expectation and the square of the coefficient of variation of the corresponding random variables using Formula (2),

$$\lambda^{-1} = \frac{a_{12}+a_{13}-\alpha a_{13}+a_{21}+\alpha a_{23}}{a_{12}a_{23}+a_{13}(a_{21}+a_{23})},$$
$$CV_a^2 = \frac{a_{12}^2-(\alpha^2-1)a_{13}^2+2a_{12}(a_{13}+a_{21})+2\alpha(\alpha-1)a_{13}a_{23}+(a_{21}+2a_{23}-\alpha a_{23})(a_{21}+\alpha a_{23})}{(a_{12}+a_{13}-\alpha a_{13}+a_{21}+\alpha a_{23})^2}.$$

   For b and CV_s^2, we obtain similar relations by replacing α by β and a_{ij} by b_{ij}.
3. Initialization of parameters of other time distributions of T is performed depending on the moments defined above. For the gamma distribution ($\mathcal{G}$) with parameters (γ, δ) and density function

$$f_T(x) = \frac{\delta(\delta x)^{\gamma-1}e^{-\delta x}}{\Gamma(\gamma)},\ x \geq 0,$$

the parameters are calculated from the relations

$$\gamma = \frac{1}{C_T^2}, \delta = \frac{\gamma}{\mathbb{E}[T]}.$$

For a Pareto distribution ($\mathcal{PR}$) with parameters (γ, δ) and density function

$$f_T(x) = \gamma \delta^\gamma x^{-\gamma-1}, \; x \geq \delta,$$

the parameters depend on the moments of the random variable T in accordance with the relations

$$\gamma = 1 + \frac{\sqrt{1+C_T^2}}{C_T}, \delta = \frac{\gamma - 1}{\gamma}\mathbb{E}[T].$$

Finally, for a lognormal distribution ($\mathcal{LN}$) with parameters (m, σ) and density function

$$f_T(x) = \frac{1}{\sigma x}\Phi\Big(\frac{\log(x) - m}{\sigma}\Big), \; x \geq 0,$$

the parameters are defined as

$$\sigma = \sqrt{\log(1+C_T^2)}, \; m = \log(\mathbb{E}[T]) - \frac{\sigma^2}{2}.$$

4. We calculate the average number of customers in the system using the approximation (3) and the simulation models proposed above for systems $PH/PH/10$, $\mathcal{G}/\mathcal{G}/10, \mathcal{PR}/\mathcal{PR}/10, \mathcal{LN}/\mathcal{LN}/10$. Here, in principle, any combination of distributions could be used. We illustrate the computational results in form of pairs $(\rho, \mathbb{E})$ in Figure 2. The evaluation of the load factor of the system is based on the values of parameters for arrival and service processes generated in step 1. As we can see, the curves match to a large extent. Some deviations are observed as the system load increases, e.g., for $\rho > 0.9$, especially for the Pareto distribution. This is due to the increase in the variance of the estimate of the average number of customers in the system at higher loads. In addition, the Pareto distribution has heavy tails, due to which the system can be characterized by slower growth of the number of customers in the system at the same load compared to other distributions. Thus, the choice of the first two moments of the interarrival and the the service times as characteristic features for estimating the average metrics of the given queueing system is quite justified.

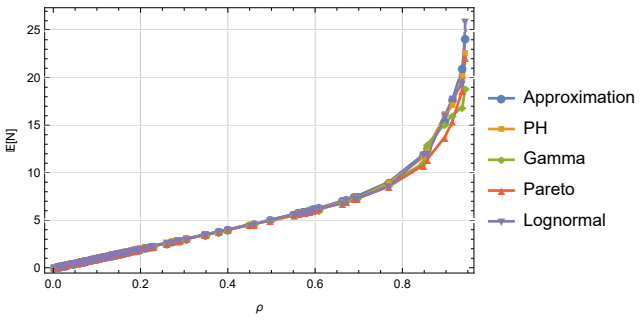

Figure 2. Comparison of approximation and simulation model results for different distributions.

5. The final part in the validation process of the simulation models is the comparison of the two algorithms, discrete-event and by the departure moments, with the exact values obtained for the $M/M/K$ queueing system. Figure 3 shows the results of

calculations of the average number of customers in the system $M/M/10$ depending on the load factor ρ using simulation models and explicit formula

$$\mathbb{E}[N] = K\rho + \frac{(K\rho)^{K+1}/(K!K(1-\rho)^2)}{\sum_{n=0}^{K-1}(K\rho)^n/n! + (K\rho)^K/(K!(1-\rho))}.$$

As we can see, the graphs are indistinguishable up to a certain value of ρ, and, only at high load, small deviations are observed due to the large variance of the obtained estimates. When simulating queues in heavy traffic, estimators of mean characteristics converge slowly to their true values. This problem is exacerbated when the distributions of interarrival and service times have heavy tails.

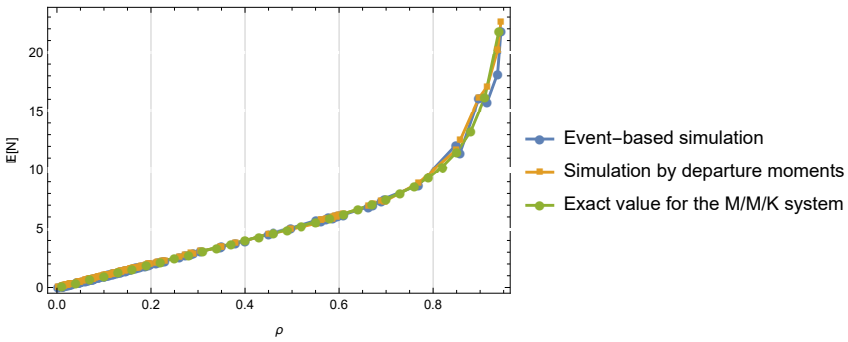

Figure 3. Comparison of calculations for two types of simulation models with exact values for the $M/M/K$ system.

In the following, we enumerate key ML use cases for performance analysis and optimization of the $GI/G/K$ queueing system.

5. Regression Tasks

In this section, we will consider some typical regression problems that usually arise in the analysis of queueing systems, for which we will use machine learning. Our goal is to obtain such trained ML algorithms that would give a qualitative estimate of the characteristics of the system of interest with a continuous domain of definition. The input features of the training and test samples for a fixed value of K, as mentioned above, are the three-dimensional map $x = (\rho, CV_a^2, CV_s^2)$. Note that the number of servers can also be made an additional feature, and the data for different numbers of servers can be accumulated accordingly. We decided to simplify the problem to speed up the data generation and the training speed of the ML algorithms, and we will investigate a particular case for $K = 10$.

Evaluation of the quality of ML algorithms is an integral part of any machine learning project. The following performance metrics are used for the regression problem. The mean square error

$$MSE_L = \frac{1}{|D_L|} \sum_{\substack{j=1 \\ y_j \in D_L}}^{|D_L|} (y_j - \hat{y}_j)^2 \tag{4}$$

takes the mean of the square of the difference between the true and predicted values. The advantage of MSE_L is that it is easier to compute the gradient in this way. If the square of the error is taken instead of the difference, the effect of larger errors becomes more pronounced compared to smaller errors, so the model may focus more on larger errors.

The quality of the regression model estimation is also investigated using the coefficient of determination R^2, defined as

$$R^2 = 1 - \frac{MSE_L}{S_L^2}, \quad S_L^2 = \frac{1}{|D_L|}\sum_{j=1}^{|D_L|}(y_i - \bar{y}_L)^2, \tag{5}$$

where S_L^2 denotes the sample variance on the training set. This characteristic represents the fraction of the variance determined by the selected model.

5.1. Estimation of the Average Number of Customers in the System

Calculation of system- and customer-level average performance metrics plays a key role by analyzing the queueing system's efficiency. To estimate the average number of customers in the system, we generate a set of values of PH distribution parameters for the interarrival and service processes. To compute the target function $y = \mathbb{E}[N]$, we use a faster simulation Algorithm 2 based on departure moments. We obtain a set D of observed pairs for the training D_L and test D_T samples,

$$D = D_L \cup D_T = \{(\rho^{(j)}, (CV_a^2)^{(j)}, (CV_s^2)^{(j)}) \to \mathbb{E}[N]^{(j)} : j = 1, \ldots, |D|\}$$

with the length $|D|$. Note that when choosing transition rates for the PH distributions of the interval $[0, 2]$, we most often obtain the set of parameter values at which $\rho < 0.95$. Therefore, in order to have enough observations in the case of heavy traffic, we created two separate samples of size $|D| = 200,000$ for $0 < \rho < 0.95$ and $|D| = 2000$ for $0.95 \leq \rho < 1$. The squares of the coefficients of variation for the first sample take values of

$$0.5 < CV_a^2 < 19.3,\ 0.5 < CV_s^2 < 10.3,$$

and for the second sample

$$0.5 < CV_a^2 < 3.5,\ 0.5 < CV_s^2 < 5.5.$$

Of course, these intervals do not cover all theoretically possible moments' values. If for real data the values of the feature vector are outside the definition area which was used to train the ML algorithms, the values of the estimates may contain significant errors. Here we wanted to show exactly the principle of using supervised ML algorithms for an arbitrary type of queueing system. Therefore, if there is a need to train the algorithms on data outside these intervals, one can change the values of the parameters for PH distributions, as well as the dimensionality of their representations, to obtain the required intervals for the feature vector used.

Next, we divide the available samples into training and test parts in a ratio of 70% and 30%, respectively.

The test results of six trained ML regression algorithms are collected in Table 1 for sample $0 < \rho < 0.95$ (a) and $0.95 \leq \rho < 1$ (b). The supervised ML methods used are: nearest neighbor, decision tree, random forest, gradient boosted tree, Gaussian regression, and neural network. The table shows the square root of the mean square error, i.e., $\sigma = \sqrt{MSE_L}$, and the coefficient of determination R^2. As can be seen in the tables, the ML algorithms produce better estimates of the data for $0 < \rho < 0.95$ where there are not too many outliers. The neural network with a value of $R^2 = 0.996$ showed the highest efficiency. Cross-validation was used to assess how the results obtained can be generalized to an independent data set. The analysis showed that trained algorithms can ensure the high quality of estimates of the target function for unseen data. Table 2 collects verifica-

tion data through metrics σ and R^2 of the trained neural network for queueing systems having certain parametric distributions of the interarrival and service times. Data of size 100 were generated for each combination of the specified distributions under condition $0 < \rho \leq 0.95$. As can be seen, the values of these metrics correspond to the result obtained in the verification of the model with PH distributions.

Table 1. The quality metrics of estimates by different ML methods for the sample $0 < \rho < 0.95$ (**a**) and $0.95 \leq \rho < 1$ (**b**).

(a)			**(b)**		
Method	σ	R^2	**Method**	σ	R^2
Nearest Neighbors	0.147	0.990	Nearest Neighbors	5.860	0.765
Decision Tree	0.111	0.994	Decision Tree	6.020	0.751
Random Forest	0.447	0.903	Random Forest	5.770	0.772
Gradient Boosted Tree	0.253	0.969	Gradient Boosted Tree	5.690	0.778
Gauss Regression.	0.443	0.909	Gauss Regression	5.650	0.781
Neural Network	0.088	0.996	Neural Network	5.780	0.771

At the same time, for the sample corresponding to high system load, the efficiency of all algorithms decreased significantly. The best result was shown by Gaussian Regression with the value $R^2 = 0.781$. Improving the quality of the estimation of average performance measures in heavy-traffic conditions can be challenging due to increased variability and the longer simulation times required. Some of the possible strategies which can be implemented to enhance the accuracy and efficiency of the simulator include the following: increasing simulation run time to ensure that the system reaches a steady state and to reduce the impact of initial transient conditions; discarding data from the initial transient period to avoid bias in the estimates; dividing the simulation run into several batches and calculate the average performance measure based on these batches; using overlapping batches to improve the estimations; using a control variable, for example, the average performance measure of the equivalent Markov queueing system with known theoretical results, that is correlated with the estimated measure. In addition, a possible solution to the problem is to generate a significant amount of data for individual intervals of small size, where the ρ takes values such as $0.95 < \rho \leq 0.96, 0, 96 < \rho \leq 0.97, \ldots$, and then train the ML algorithms for each interval separately.

Table 2. Verification of the neural network for the queuing system with different interarrival and service time distributions by metrics σ and R^2.

Service \ **Arrival**	$\mathcal{PH}$	$\mathcal{G}$	$\mathcal{PR}$	$\mathcal{LN}$
$\mathcal{PH}$	$\sigma = 0.088$ $R^2 = 0.993$	$\sigma = 0.314$ $R^2 = 0.981$	$\sigma = 0.151$ $R^2 = 0.965$	$\sigma = 0.407$ $R^2 = 0.973$
$\mathcal{G}$	$\sigma = 0.302$ $R^2 = 0.982$	$\sigma = 0.415$ $R^2 = 0.973$	$\sigma = 0.017$ $R^2 = 0.995$	$\sigma = 2.401$ $R^2 = 0.902$
$\mathcal{PR}$	$\sigma = 0.284$ $R^2 = 0.967$	$\sigma = 0.369$ $R^2 = 0.954$	$\sigma = 0.379$ $R^2 = 0.940$	$\sigma = 0.998$ $R^2 = 0.934$
$\mathcal{LN}$	$\sigma = 0.169$ $R^2 = 0.986$	$\sigma = 0.928$ $R^2 = 0.937$	$\sigma = 0.254$ $R^2 = 0.949$	$\sigma = 2.221$ $R^2 = 0.924$

The scatter plots for the best algorithms to estimate the average number of customers in the system for two intervals of load factor of the system are shown in Figure 4. As can be seen in the plots, the scatter relative to the ideal prediction increases as the number of customers in the system increases, which is a consequence of the greater variance in

the estimate for the average number of customers in the system as the load increases. To improve the performance of the ML algorithm in the case of heavy traffic, the ρ values could be split into small intervals, and then the data from each subinterval could be used to train a separate ML algorithm. Also, one could try here to perform preprocessing of the input features of the sample by normalizing them.

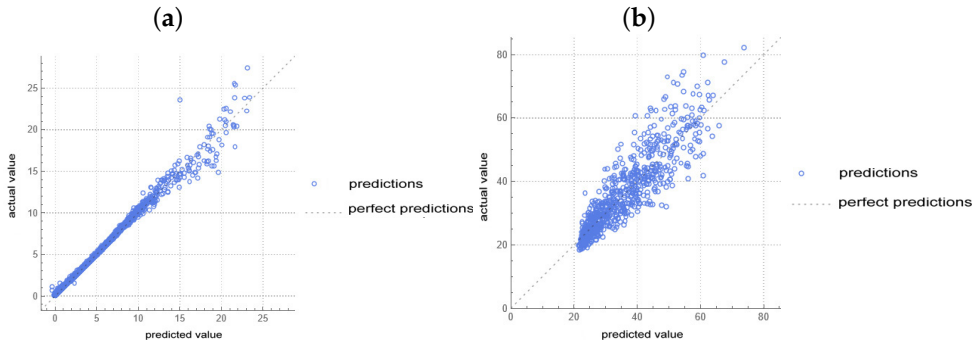

Figure 4. Scatter plots for predicting the average number of customers in the system with the load factor (**a**) $0 < \rho < 0.95$ and (**b**) $0.95 \leq \rho < 1$.

Figure 5 shows the plots of empirical cumulative distribution functions (CDFs) for the absolute error using ML algorithms.

For the sample $\rho < 0.95$, all algorithms show that the absolute error is almost certainly less than 1. The random forest method is slightly inferior to all other algorithms. For high system load, when $0.95 \leq \rho < 1$, all algorithms also show similar results; that is, no significant advantage of one of the methods is observed. The probability that the deviation from the simulation value is less than 5 is about 0.75, and about 0.90 for less than 10. However, the number of customers in the system is large at this high load, so we also consider this result acceptable.

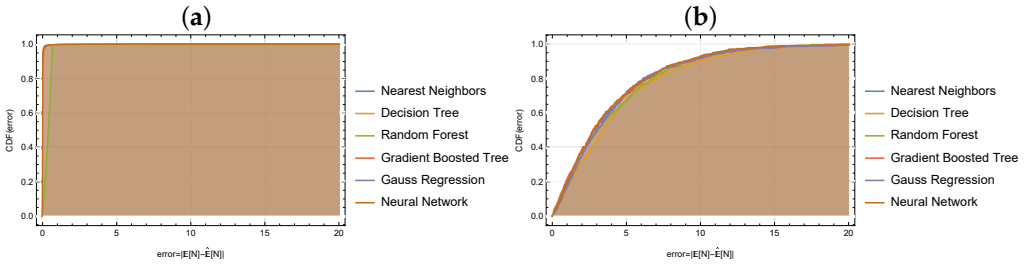

Figure 5. Empirical distribution functions of the absolute error (**a**) $0 < \rho < 0.95$ and (**b**) $0.95 \leq \rho < 1$.

In addition to the average number of customers in the system, the approach proposed in this section can also be used to estimate other average characteristics such as the average number of customers in the queue, the average number of busy servers, the average duration of the busy period, the probability of blocking a customer, etc. For this purpose, appropriate samples need to be generated and ML algorithms need to be trained. Verification of the results can be performed using available numerical calculations on the simulator, available approximations, or by means of analytical relations obtained for a simplified model, usually with exponential distributions of time intervals. Thus, ML offers a promising approach for approximating or predicting the mean performance metrics of queueing systems using data-driven models.

5.2. Estimation of the Distribution for the Number of Customers in the System

In this section, we describe the procedure for estimating the stationary distribution of the number of customers in the system. As is well known, there are only approximate

solutions for this characteristic concerning the queueing system $GI/G/K$. As shown in [6], the stationary probabilities $\pi_j = \mathbb{P}[N_K = j]$ of the number of N_K customers in a system with K servers is determined by the relation

$$\pi_j = (1 - P_d) f_1(j) + P_d f_2(j), \tag{6}$$

where $P_d = \sum_{j=K}^{\infty} \pi_j$ is the probability of blocking a customer, which is found by using the ratio

$$P_d = \rho \frac{\mathbb{E}[N_K] - c\rho}{\mathbb{E}[N_1] - \rho}.$$

Here, the mean values of $\mathbb{E}[N_K]$ and $\mathbb{E}[N_1]$, which stand, respectively, for the system with K servers and with one server, are determined by Formula (3). The functions $f_1(j)$, $j = 0, 1, \ldots, K-1$ and $f_2(j), j = K, K+1, \ldots$ are conditional probabilities for the random variable N_K, respectively, for states where there is at least one idle server and when all servers are occupied. These probabilities are approximated by the following expressions:

$$f_1(j) = \rho_1^j \frac{(1-\rho_1)^{K-j}}{1-\rho_1^K}, \; j \le K-1,$$
$$f_2(j) = (1-\rho_2)\rho_2^{j-K}, \; j = K, K+1, \ldots.$$

The auxiliary coefficients ρ_1 and ρ_2 are calculated, in turn, using the following system of equations:

$$(1-P_d)\frac{K\rho_1 - K\rho_1^K}{1-\rho_1^K} + P_d\Big(\frac{\rho_2}{1-\rho_1} + K\Big) = \mathbb{E}[N_K],$$
$$P_d \frac{\rho_2}{1-\rho_2} + K\rho = \mathbb{E}[N_K].$$

Formula (6) will be used to perform a comparative analysis of discrete value density distribution functions, which we will approximate using continuous functions.

Our task is to use ML algorithms for an alternative possibility of estimating the stationary distribution of the number of customers in the system. As input feature x in our sample, we again use, as in the previous section, three components: the system load factor ρ, the coefficients of variation CV_a^2, and CV_s^2. The question is what we will use as the target function y. In fact, we are interested in the probabilities of system states. Using the simulation model, we can compute $\pi_j, j = 0, 1, \ldots$. For this problem, we use Algorithm 1 of the event-based simulation, in which there is the possibility of recording the system's visit to a state and recording the corresponding holding time. In the algorithm, we need to initialize a separate variable $H_x = 0$ of a holding time for each state x to be recorded, and then, at line 6, add the entry

$$H_x \leftarrow H_x + T_x.$$

At the end of the program cycle, the result of the approximate calculation of the state probabilities $\pi_x \approx \frac{H_x}{T_s}$ is displayed. We will calculate these probabilities, for example, up to the system state, which corresponds to the number of customers in the system equal to 50. This number of points should be enough to accurately determine the shape of the distribution.

We would like to reduce the dimensionality of our data and describe the target function in a more compact way, since a vector of dimensionality 51 in the output will clearly create difficulties in training ML algorithms. For this purpose, we performed many experiments to compute the density distribution of the number of customers in the system and to evaluate a

suitable continuous parametric function. This function should have the properties of density of distribution and depend on as few parameters as possible for the convenience of their further estimation from the available set of state probability values. After careful analysis, we conclude that the most universal suitable continuous function is the density of the generalized Weibull distribution (see, e.g., [52]), where we fix the scaling parameter λ and set it equal to 1,

$$f(x;\gamma,\delta) = \gamma\delta x^{\delta-1}(1-e^{-x^{\delta}})^{\gamma-1}e^{-x^{\delta}}. \tag{7}$$

Here, γ and δ are the form parameters. Thus, data of the following structure are generated:

$$D = D_L \cup D_T = \{(\rho^{(j)}, (CV_a^2)^{(j)}, (CV_s^2)^{(j)}) \to (\gamma^{(j)}, \delta^{(j)}) : j = 1, \ldots, |D|\},$$

where $|D| = 45,000$ and the target function y is given here by a two-dimensional vector of estimated parameters obtained by the least square method for the generalized Weibull distribution.

Since the target function is given by a vector, we use a neural network to solve the regression problem. The test of the trained algorithm on the test sample showed the following results: $\sigma = \sqrt{MSE_L} = 1.368$ and $R^2 = 0.854$. Note that the coefficient of determination R^2 in the case of a multivariate target function is calculated in the same way as in the univariate case (see (5)), with the only being difference that the formula calculates the accumulated sums for all components of the target function according to its dimensionality.

In Figure 6, a comparison example of graphs for the continuous probability density function (PDF) $f(x;\gamma,\delta)$ obtained by simulation, trained neural network, and by means of the approximate state probability distribution (6) is presented for two different sets of parameters corresponding to the varied system load. It can be seen that the functions are quite close to each other, with a slight overestimation being given by the neural network at the peak. However, in general, by the shape of the functions and the obtained estimates of the parameters of the continuous densities, we can conclude that the proposed approach is quite efficient. Finally, the continuous PDF $f(x;\gamma,\delta)$ with the estimated parameters can be used to compute the state probability using a continuity correction,

$$\pi_j = \mathbb{P}[X_K = j] = \mathbb{P}\Big[j - \frac{1}{2} \le X_K \le j + \frac{1}{2}\Big] \approx \int_{j-1/2}^{j+1/2} f(x;\gamma,\delta)dx.$$

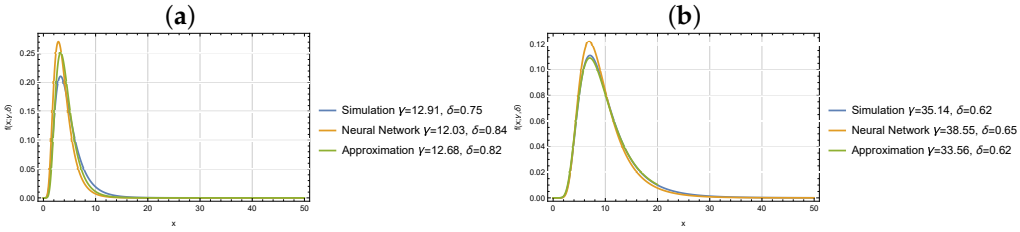

Figure 6. Comparison of continuous density distribution functions (**a**) $\rho = 0.41$, $C_a^2 = 0.97$, $C_s^2 = 1.11$ and (**b**) $\rho = 0.82$, $C_a^2 = 0.93$, $C_s^2 = 1.11$.

5.3. Estimation of the Waiting and Sojourn Time Distributions

The waiting time of a customer in the queue and the sojourn time a customer spends in the system are very interesting and informative characteristics of any queueing system. The sojourn time of a customer in the system is equal to the sum of waiting time and service time. The average waiting time $\mathbb{E}[W]$ and the sojourn time $\mathbb{E}[T]$ can be calculated by applying Little's formula, $\mathbb{E}[T] = \frac{\mathbb{E}[N]}{\lambda}$, $\mathbb{E}[W] = \mathbb{E}[T] - b$ if the average number of customers in the system is calculated. The Section 5.1 is devoted to the estimation of this characteristic. Here, we will be interested in estimating the stationary distributions of

waiting and sojourn times using the same input feature x used earlier. The question is how we need to describe a target function y that closely approximates the distribution functions $F_W(t) = \mathbb{P}[W < t]$ and $F_T(t) = \mathbb{P}[T < t]$ of the continuous random variables of waiting time W and sojourn time T. We propose using the simulation model to estimate the moments of random variables W and T, namely the first three moments, and then use them to fit the parameters of PH distributions with two transition states [12], which will approximate the desired functions $F_W(t)$ and $F_T(t)$. We can also estimate the first five moments. In this case, we obtain the parameters of the PH distribution with three transition states. We generate the data as a set

$$\begin{aligned} D = &\{(\rho^{(j)}, (CV_a^2)^{(j)}, (CV_s^2)^{(j)}) \rightarrow \\ &(\mathbb{E}[W]^{(j)}, \mathbb{E}[W^2]^{(j)}, \mathbb{E}[W^3]^{(j)}, \mathbb{E}[T]^{(j)}, \mathbb{E}[T^2]^{(j)}, \mathbb{E}[T^3]^{(j)}) : j = 1, \ldots, |D|\}, \end{aligned}$$

where $|D| = 60,000$. Event-based simulation should be used to obtain samples of waiting and sojourn times. It is required to capture all waiting times that occur. For this purpose, two variables are introduced to monitor the waiting time whenever an incoming customer joins the queue and the sojourn time when the customer enters the system. In the case where an incoming customer immediately arrives to a free server, it is necessary to record the waiting time 0, and we record the time of servicing the customer as the sojourn time on the server.

The difficulty in recording the waiting and sojourn times is that there may be more than one customer in the queue at a given time. Therefore, in the event-base simulation, we define two types of arrays: $W = \{\}$ and $T = \{\}$ for accumulated values; and $W_m = \{\}$ and $T_m = \{\}$ to store current values of waiting and sojourn times. All of these arrays are initialized as empty sets. Thus, whenever a new customer event is logged, a check is made first to see if a server is idle. If one of the servers is idle, a 0 is written in the W_m data array; that is, $W_m \leftarrow \{W_m, 0\}$, and the service time $T_m \leftarrow \{T_m, x_b\}$, is written. Otherwise, we redefine $W \leftarrow \{W, 0\}$ with the addition of entry 0, since the sojourn time has not yet accumulated until this customer arrives, and $T \leftarrow \{T, x_b\}$. When we move on to the next event, the sojourn time T_x is added to all entries W and T, $W \leftarrow W + T_x, T \leftarrow T + T_x$. If a customer leaves the queue after the end of the service, we transfer the contents of the accumulated waiting time to the waiting time sample $W_m \leftarrow \{W_m, \max\{W\}\}$ and remove the corresponding entry from the array W; that is, $W \leftarrow \{W \setminus \{\max\{W\}\}\}$. We perform similar manipulations with the arrays T_m and T. When the simulation is complete, we have samples of waiting and sojourn times, which are then used to compute empirical moments.

Since the target function in our data is represented as vectors of three waiting and sojourn times moments, a multilayer neural network is used to solve the regression problem. Verifying the trained network in a test sample gives an accuracy of the order of $R^2 = 0.691$. To improve accuracy, separate models could be used with the scalar values of each moment as the target function. We could also use a data preprocessing step using log-transformation, since, as the moment order increases, the values of these moments grow exponentially. In the following experiment, we take the following records from the test sample for two different values of the ρ system load:

$$\begin{aligned} &(0.409, 1.154, 1.136) \rightarrow (0.009, 0.012, 0.019, 4.149, 38.094, 530.223), \\ &(0.937, 0.998, 0.992) \rightarrow (6.293, 89.586, 1682, 13.354, 265.413, 6687.48). \end{aligned}$$

The trained neural network produces the following estimates for the moments of the random variables W and T:

$$(0.409, 1.154, 1.136) \to (0.010, 0.013, 0.026, 3.912, 31.948, 696.326),$$
$$(0.937, 0.998, 0.992) \to (6.032, 60.372, 850, 13.290, 276.750, 7779.935).$$

Now, estimating the values of the parameters of the PH distributions from the moment estimates, we obtain the following approximations for $\rho = 0.409$:

$$F_W(t) \approx 1 - \alpha_W e^{S_W t} e,\ \alpha_W = (0.012, 0.988),\ S_W = \begin{pmatrix} -1.413 & 1.413 \\ 0 & -1039.58 \end{pmatrix},$$
$$F_T(t) \approx 1 - \alpha_T e^{S_T t} e,\ \alpha_T = (0.0001, 0.9999),\ S_T = \begin{pmatrix} -0.013 & 0.013 \\ 0 & -0.256 \end{pmatrix}$$

and for $\rho = 0.937$,

$$F_W(t) \approx 1 - \alpha_W e^{S_W t} e,\ \alpha_W = (0.012, 0.988),\ S_W = \begin{pmatrix} -1.413 & 1.413 \\ 0 & -1039.58 \end{pmatrix},$$
$$F_T(t) \approx 1 - \alpha_T e^{S_T t} e,\ \alpha_T = (0.883, 0.117),\ S_T = \begin{pmatrix} -0.133 & 0.133 \\ 0 & -0.151 \end{pmatrix}.$$

A PH distribution with two transition probabilities is not always an adequate model for approximating the functions $F_W(t)$ and $F_T(t)$. For certain sets of moments, there may simply not be suitable values of the PH distribution parameters. In this case, it is necessary to increase the dimensionality of the PH distributions. Alternatively, one can search for other parametric distributions that approximate the computed functions well. For example, for $F_W(t)$ one can use a distribution from the exponential class, such as

$$F_W(t) \approx 1 - \gamma e^{-\delta t}.$$

This distribution gives us the possibility to also estimate the probability $\mathbb{P}[W > 0]$ of system blocking; that is, a non-zero waiting time. Statistical testing for the conformity of waiting time samples with the parametric distributions used confirmed the correctness of their use for our experiments. The parameters (γ, δ) are estimated using the first two moments. For $\rho = 0.409$ and $\rho = 0.937$, we obtain, respectively, $(\gamma, \delta) = (0.015, 1.565)$ and $(\gamma, \delta) = (0.884, 0.140)$.

Figure 7 shows the CDF plots $F_W(t)$ obtained empirically from a sample of random variable values, as well as by means of two approximations, namely the exponential-type distribution and the PH distribution. The parameters of these distributions were estimated using the method of moments, whose values were, in turn, calculated from the regression model of the neural network. Here, we can observe the close location of functions calculated by different methods, which confirms the effectiveness of the proposed approach.

The Weibull distribution is also suitable for approximating the function $F_T(t)$,

$$F_T(t) \approx 1 - e^{-\delta t^\gamma},$$

for which the following parameter estimates are obtained: in the case of $\rho = 0.409$, we have $(\gamma, \delta) = (0.889, 0.312)$, and for $\rho = 0.937$, $(\gamma, \delta) = (1.596, 0.013)$. Figure 8 shows the plots of the distribution function $F_T(t)$ obtained, similar to the waiting time, empirically from the sample, as well as by using two approximations: the Weibull distribution and the PH distribution. The parameters of these functions were also estimated using moments

computed by the neural network. Here again, we observe that the regression model of the neural network has proved its worth in estimating stationary distributions of continuous random variables. Note that an alternative option may be to use ML algorithms to estimate the parameters of the approximating distributions themselves, as suggested in the previous section, to estimate the probability distribution of states by means of a continuous function. In this case, it is necessary to generate data in the form of

$$D = \{(\rho^{(j)}, (CV_a^2)^{(j)}, (CV_s^2)^{(j)}) \to (\gamma_W^{(j)}, \delta_W^{(j)}, \gamma_T^{(j)}, \delta_T^{(j)}) : j = 1, \ldots, |D|\},$$

where the vector of parameters of approximating functions for random variables W and T acts as the target function. In addition, these data should be used to train the ML algorithms, which can then be applied to new input data.

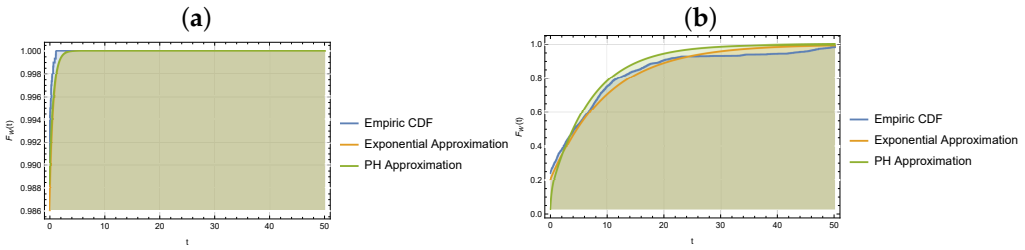

Figure 7. Comparison of distribution functions $F_W(t)$ (**a**) $\rho = 0.41$, $C_a^2 = 1.15$, $C_s^2 = 1.14$ and (**b**) $\rho = 0.94$, $C_a^2 = 0.99$, $C_s^2 = 0.99$.

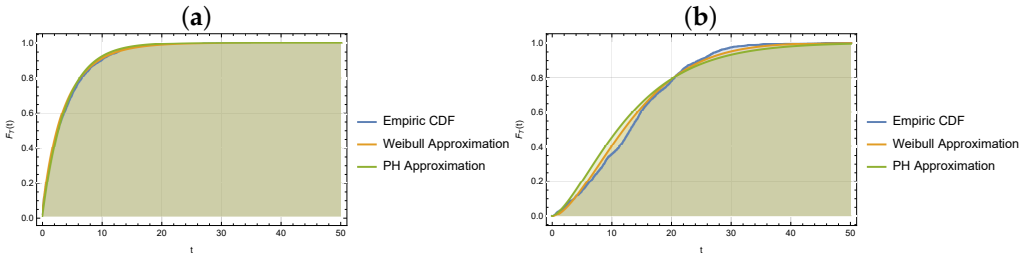

Figure 8. Comparison of distribution functions $F_T(t)$ (**a**) $\rho = 0.41$, $C_a^2 = 1.15$, $C_s^2 = 1.14$ and (**b**) $\rho = 0.94$, $C_a^2 = 0.99$, $C_s^2 = 0.99$.

The examples presented show that ML methods for regression in queueing systems can handle arbitrary distributions of arrival and service times, predict performance metrics for unseen system configurations, and provide instantaneous predictions.

6. Classification Tasks

This section presents a number of examples of the application of ML algorithms to classification problems that arise in the $GI/G/K$ system. Here, discrete labels or classes serve as the target function. The following metrics are used to evaluate the classification performance. A confusion matrix is a tool that is used to evaluate the performance of a classification model. It is a table that describes the relationship between the actual and predicted classes for a given set of data. This is useful in supervised learning where the target variable has discrete categories. Using the values in the confusion matrix, various performance metrics can be calculated; for example, the accuracy, ACC, can be obtained by taking the average of the values lying on the main diagonal of the confusion matrix. The $F1$ measure is a weighted harmonic mean between the precision and recall measures,

$$F1 = 2\frac{1}{\frac{1}{Precision} + \frac{1}{Recall}}, \tag{8}$$

where precision is defined as the ratio of the number of true positives to the number of all predicted positives and recall is defined by the ratio of the number of true positives to the number of all data points that should have been classified as positive. This metric takes values on the interval $[0, 1]$ and tells us about the accuracy of the classifier, in particular how many data points it classifies correctly, as well as its robustness, i.e., it should not miss a significant number of data points. Cross-entropy is a widely used loss function in classification problems, particularly in multiclass classification. It measures the difference between the true probability distribution (actual labels) and the predicted probability distribution produced by the model. The cross-entropy loss for a single jth data point is defined as

$$L_j = -\sum_{i=1}^{C} y_{ij} \log(\hat{y}_{ij}),$$

where y_i is a true label for class i (1 if it is the correct class, 0 otherwise) and $\hat{y}_i$ is a predicted probability for class i. The average cross-entropy is then defined as

$$Entropy = -\frac{1}{|D|} \sum_{j=1}^{|D|} L_j, \tag{9}$$

where $|D|$ is the number of examples in the data set D. The lower *Entropy* in (9) indicates that the model is assigning higher probabilities to the correct class.

6.1. Classifying by Waiting Time Threshold

There are certain situations where knowledge of a specific value of some characteristic of the system is not required, but it is necessary to determine, for example, that at given system parameters, the waiting time will be below or above some fixed threshold or within some given interval. To solve such problems, let us generate data in the form of a set

$$D = \{(\rho^{(j)}, (CV_a^2)^{(j)}, (CV_s^2)^{(j)}) \to y^{(j)} : j = 1, \ldots, |D|\},$$

where $y^{(j)} \in \{0, 1, ..., C-1\}$, C is the number of discrete labels, categories, or classes, which, in turn, are determined based on values of the average waiting time, i.e., $y^{(j)} = c$, conditional on

$$\mathbb{E}[W]^{(j)} \in (a_c; b_c],\ c = 0, 1, \ldots, C-1.$$

First, consider the binary classification problem when, for a given threshold w, class 0 is defined by the inequality $\mathbb{E}[W]^{(j)} < w$, and class 1 is defined by the inequality $\mathbb{E}[W]^{(j)} \geq w$.

In the following example, we take the values $w = 0, 4, 7$ as thresholds. For threshold $w = 0$, the input data of the system will be classified for the presence of waiting time. Table 3 presents a table of binary classification performance characteristics for the two algorithms that give the best results: logistic regression and neural network. We balanced training and test samples to ensure that the classification results are representative. Performance characteristics are the overall accuracy value, *ACC*, and the average cross-entropy. Low values of this characteristic correspond to high values of accuracy. We can observe a rather high quality of binary classification using the above algorithms, with the presence of a waiting time at the threshold $w = 0$ determined with a lower probability. This is because, in the training sample, there are a lot of values of the average waiting time lying around zero which cannot be effectively classified.

Table 3. Performance characteristics of logistic regression classifier and neural network for different thresholds.

Method	Threshold w	Accuracy ACC	Entropy
Logistic Regression	0	0.923	0.193
Neural Network	0	0.903	0.215
Logistic Regression	4	0.991	0.034
Neural Network	4	0.927	0.076
Logistic Regression	7	0.990	0.101
Neural Network	7	0.970	0.099

In Figure 9, we present the confusion matrices for the three thresholds and the two classification algorithms used. As can be seen, the matrices have a clearly dominant main diagonal, which corresponds to the results of Table 3.

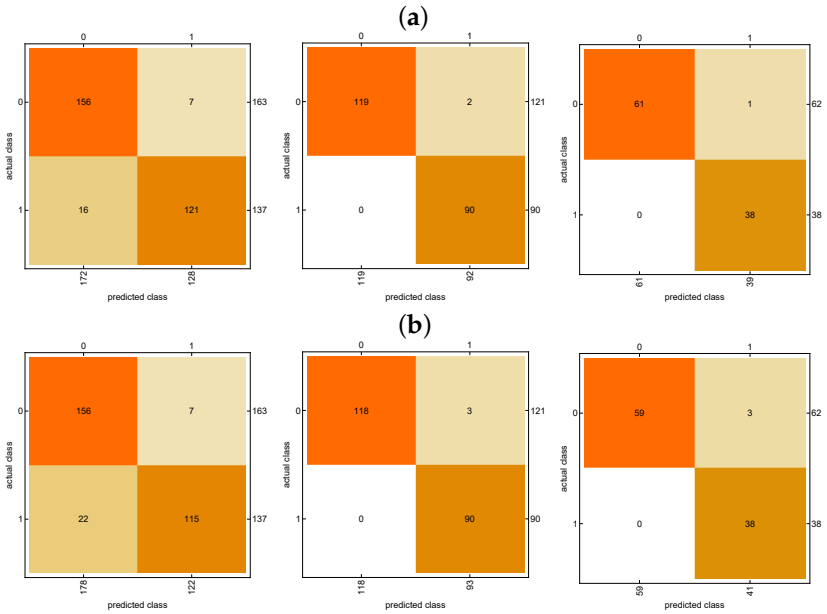

Figure 9. Comparison of confusion matrices for thresholds $w = 0, 4, 7$ for classification by (**a**) logistic regression and (**b**) neural network.

Next, let us consider an example of multiclass classification. We use the classes introduced in the previous binary classification example; namely, $\mathbb{E}[W]^{(j)} = 0$ corresponds to class 0, $0 < \mathbb{E}[W]^{(j)} \leq 4$ – class 1, $4 < \mathbb{E}[W]^{(j)} \leq 7$ – class 2, and finally, $\mathbb{E}[W]^{(j)} > 7$ – class 3. We now have $C = 4$ classes. The quality metrics of this classification problem for different ML methods are summarized in Table 4.

Table 4. Performance characteristics of different classifiers for 4-class classification.

Methods	Accuracy ACC	Entropy
Logistic Regression	0.857	0.323
Nearest Neighbor	0.847	0.367
Decision Tree	0.880	0.339
Random Forest	0.895	0.300
Gradient Boosted Tree	0.898	0.255
Support Vector Machine	0.788	0.506
Naive Bayes	0.857	0.387
Markov Model	0.698	0.824
Neural Network	0.887	0.351

Gradient boosted tree, random forest, and neural network methods proved to be the most productive here. Figure 10 shows the confusion matrices for three methods that showed the best results. As we can see, the error probability is higher for class 2 and class 3 detection. However, in general, the accuracy is almost 90%, which is a good enough result. To further improve performance, we could experiment with different samples and collect more data corresponding to the average waiting time values near the selected thresholds w. The approach proposed in this section can also be used to classify the values of any other performance and reliability characteristic of the queue, for example, to predict categorical outcomes such as whether the system is stable or overloaded.

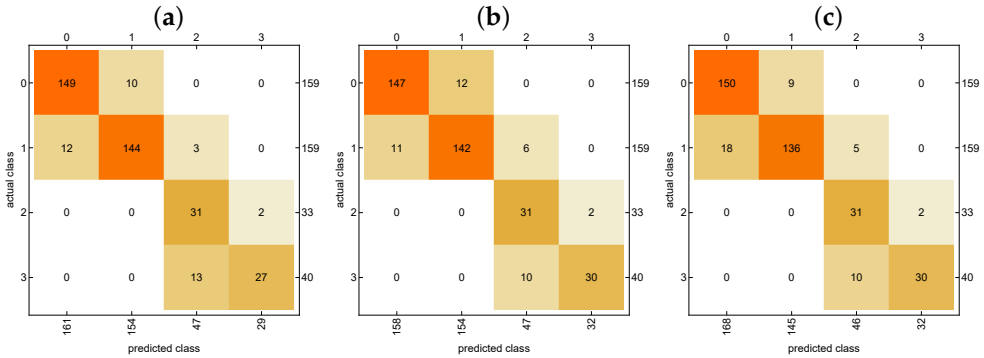

Figure 10. Comparison of confusion matrices for classification by (**a**) gradient boosted tree, (**b**) random forest, and (**c**) neural network methods.

6.2. Classification in Parametric Optimization Problem

In this section, we demonstrate the use of supervised ML in the parametric optimization problem. As an example, consider a simple problem of optimizing the number of K servers in a $GI/G/K$ system with the aim of minimizing the following loss functional

$$g(K) = c_0 K + c_1 \mathbb{E}[W],$$

where c_0 is the usage cost per unit of time for each server in the system and c_1 is a holding cost per unit of time for each waiting customer. Obviously, these two summands compete, since as K increases, the waiting time is getting higher, but the usage cost for servers increases, and vice versa. Figure 11 illustrates the average cost g for different values of K and c_1, with $c_0 = 0.1$ fixed. As we can see, the plots of the functions have an explicit minimum corresponding to the optimal number of servers for a given cost structure.

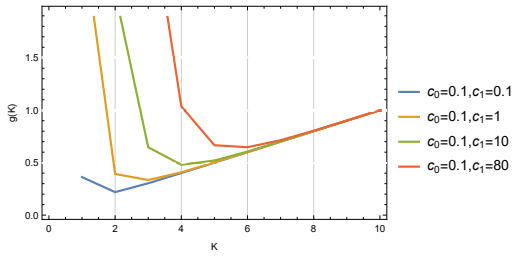

Figure 11. Values of the average cost $g(K)$ for different number of servers K, usage cost c_0 and holding cost c_1.

Our task in this section is to estimate the optimal value of K^* using classification algorithms. First, we need to generate data in the form of a set

$$D = \{(\rho^{(j)}, (CV_a^2)^{(j)}, (CV_s^2)^{(j)}, c_0^{(j)}, c_1^{(j)}) \to y^{(j)} : j = 1, \ldots, |D|\},$$

where $y^{(j)} = K^*$ and $|D| = 20,000$. The parameters of the PH distributions of the arrival and service processes are chosen randomly from the same intervals as before, but with the condition that the inequalities $\rho = \frac{\lambda b}{K} < 1$ for all $K \in \{1, 2, \ldots, 10\}$ are met. For the parameters c_0 and c_1, the following conditions are satisfied,

$$0.1 \leq c_0 \leq 10, \;\; 0.1 \leq c_1 \leq 80.$$

The numerical examples indicate that the parameter values of the given intervals correspond to possible optimal solutions K^* in the interval starting from 1 to 7; thus, it is a matter of classifying the data into the seven available classes. Table 5 collects the test results of nine trained ML classification algorithms for the optimal number of servers estimation problem. As can be seen, the nearest neighbor and the random forest methods offer the best result, with accuracy equal to almost 85%. Note that the neural network, although it gives quite acceptable results, the accuracy is slightly higher than 74%, nevertheless this method is not the best classifier for this problem. The Markov Model is not suitable here at all, because it is important to obtain the sequence of appearance of certain classes, which is not relevant for this problem, but it makes sense for tasks related to the classification of time series.

Table 5. Performance characteristics of different classifiers for the optimization problem.

Method	Accuracy *ACC*	Entropy
Logistic Regression	0.612	0.865
Nearest Neighbor	0.847	0.465
Decision Tree	0.817	0.566
Random Forest	0.845	0.393
Gradient Boosted Tree	0.840	0.484
Support Vector Machine	0.716	0.810
Naive Bayes	0.668	0.760
Markov Model	0.527	3.350
Neural Network	0.742	0.548

The confusion matrix for the nearest neighbor method, which showed the best classification result, is shown in Figure 12.

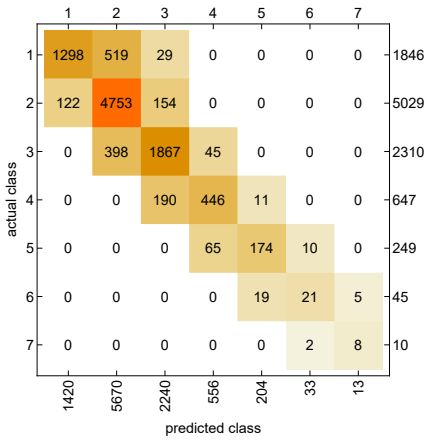

Figure 12. Confusion matrix for classification by nearest neighbor method.

This matrix has a pronounced tridiagonal structure with a dominant main diagonal. This indicates that neighboring solutions $K^* \pm 1$ give rather close results for the average cost value. Thus, these neighboring solutions for the number of servers can in principle be taken as suboptimal solutions.

6.3. Classification of Queueing Systems by Time Series

In this section, we present one more variant of the possible application of ML algorithms for solving queueing theory problems. In queueing models, the input parameters are often unknown in advance and may change with time. Assume that only the number of customers in the system dependent on time is observable. In this case, it is necessary to classify the segments of this data set by a given time series that represents the trajectory of some piecewise stationary random process. This task may be associated, for example, with detection of change points of the time series, where the parameters of the queueing systems change, or with finding some anomalous areas, in connection, for example, with DDoS hacker attacks on the server or computer system. There are a large number of various parametric and non-parametric methods for time series classification. In this section, we demonstrate the capabilities of ML algorithms for the classification of series describing the realization of some arbitrary multichannel queueing system.

Next, we discuss the formulation of specific problems and data generation. As before, it is assumed that a $GI/G/K$ system is given in general, where the distributions of time between arrivals of customers and the distribution of service times are approximated by PH distributions. For this system, we need to classify the synthetic time series shown in Figure 13, describing the number of customers $X(t)$ at time t.

This time series is obtained by combining three series generated by simulation of the queueing system $PH/PH/10$ for different load factors of the system. In Figure 13, $\rho = 0.5$ for $0 < t < t_1$, $\rho = 0.95$ for $t_1 \le t < t_2$, $\rho = 0.3$ for $t \ge t_2$, and vertical lines are also shown at the locations of the change points, $t_1 = 287$ and $t_2 = 513$, where changes occur in the arrival and service processes of the system. The values of the system parameters are chosen so that the load factor is equal to three different values. According to this criterion, we define the following classes:

- Class 1: $\rho = 0.95$,
$$\alpha = (0.078, 0.922), A = \begin{pmatrix} -2.265 & 1.772 \\ 1.440 & -2.453 \end{pmatrix}, \beta = (0.179, 0.921), B = \begin{pmatrix} -0.943 & 0.862 \\ 0.840 & -1.807 \end{pmatrix}.$$
- Class 2: $\rho = 0.5$
$$\alpha = (0.981, 0.019), A = \begin{pmatrix} -0.921 & 0.229 \\ 0.678 & -2.552 \end{pmatrix}, \beta = (0.783, 0.217), B = \begin{pmatrix} -0.415 & 0.236 \\ 1.642 & -1.678 \end{pmatrix}.$$
- Class 3: $\rho = 0.3$
$$\alpha = (0.911, 0.089), A = \begin{pmatrix} -1.942 & 0.337 \\ 1.889 & -3.055 \end{pmatrix}, \beta = (0.535, 0.465), B = \begin{pmatrix} -2.161 & 1.804 \\ 1.826 & -2.516 \end{pmatrix}.$$

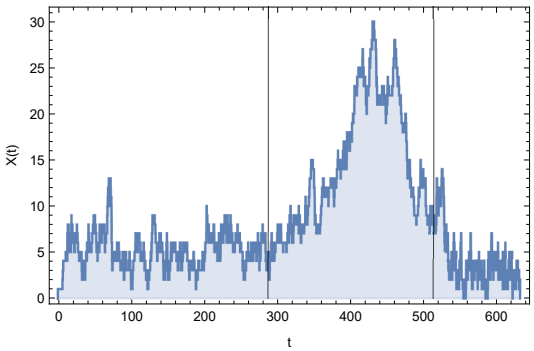

Figure 13. Time series of the number of customers with two change points $t_1 = 287$ and $t_2 = 513$.

Next, we show that trained ML algorithms cope well with the task of estimating change points t_1 and t_2. Although this task seems simple, it can be used to solve more extensive classification problems where there are a large number of classes, and more complex criteria are used to define them.

To train ML algorithms, we use simulation for three types of system parameters. In each case, the calculation is performed until the end of service of 10,000 customers. For the calculations, we use data after 1000 customers are received to remove the influence of the initial state. The resulting series are split into partial series, each containing 20 events. Next, we define the features for training the ML algorithms:

- ρ—system load factor, estimated as the average number of occupied servers, i.e., $\frac{\bar{N}-\bar{Q}}{K}$;
- λ—arrival rate, estimated as the number of incoming customers per unit of a given time interval;
- ΔN—rate of change of the number of customers in the system at the corresponding time interval;
- Δt—length of the partial series interval containing 20 events.

Figure 14 shows from left to right the pathmark of the process for the number of customers in the system for three possible states according to the value of the load factor of the system. According to the hidden Markov model, the time series in blue corresponds to $\rho = 0.5$, in red – to $\rho = 0.95$, and in green – to $\rho = 0.3$. Here, only estimates of the load factor ρ were used as observed attributes.

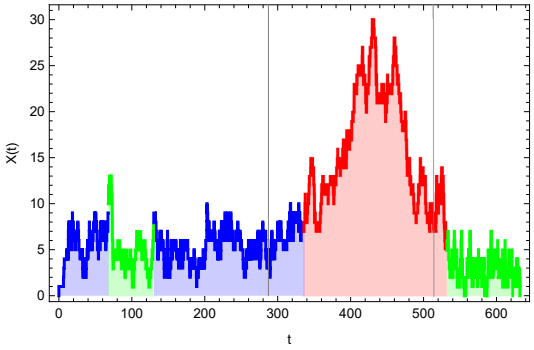

Figure 14. Time series of the number of customers in the $GI/G/10$ queueing system with real and estimated segments for $\rho = 0.5, 0.95, 0.3$ marked in blue, red and green.

The parameters of the hidden Markov model with a normal distribution for the values of the observed features are obtained from the training sample

$$\boldsymbol{\pi} = (0.393, 0.414, 0.193),\ A = \begin{pmatrix} 0.532 & 0.134 & 0.334 \\ 0.344 & 0.521 & 0.135 \\ 0.128 & 0.237 & 0.635 \end{pmatrix},$$

$$\mathbf{x}_n|B_1 \sim \mathcal{N}(0.963, 0.103),\ \mathbf{x}_n|B_2 \sim \mathcal{N}(0.516, 0.173),\ \mathbf{x}_n|B_3 \sim \mathcal{N}(0.325, 0.127).$$

The classification results are satisfactory; although, as we see, false classification is also not excluded.

In addition, we used all the observed features mentioned above to train alternative machine learning methods used for classification tasks. Based on the results of applying the trained algorithms in the path of the test process shown in Figure 13, we obtained the result that the nearest neighbor method gives the best classification accuracy $ACC = 0.92$. The confusion matrix, shown in Figure 15, obviously has a dominant main diagonal. In this case, precision and recall take the values $Precision = (0.76; 1.00; 0.94)$ and $Recall = (0.93; 0.86; 1.00)$ for classes 1, 2, and 3, respectively, and the measure $F1 = (0.84; 0.92; 0.97)$.

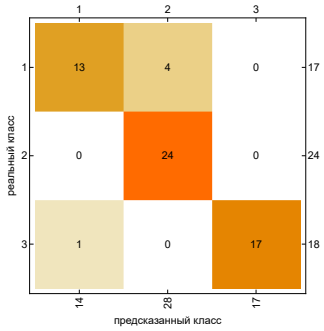

Figure 15. Confusion matrix for classification of queueing systems' parameters by nearest neighbor method.

According to the trained nearest neighbor algorithm, as can be seen from the confusion matrix, the first 28 partial rows are identified as belonging to class 2, the next 14 rows belong to class 1, and then 17 partial rows of the last group are classified as class 3. Note that although some partial rows were misclassified several times, for example, four rows of class 1 were identified as class 2 and one row of class 3 was identified as class 1, this nevertheless had almost no effect on the precision of the estimation of change points in the values of the queueing system parameters, which can be clearly seen in Figure 16. Identification of change points as boundaries of classes shows values $t_1^* = 289$ and $t_2^* = 514$, marked as a red vertical dashed line, which practically coincide with the time thresholds initially set. Note that classification tasks can also be solved using unsupervised ML, e.g., by using the cumulative sum charts (CUSUM) change point detection algorithm, for which, as mentioned above, there is no need to specify the values of the target function.

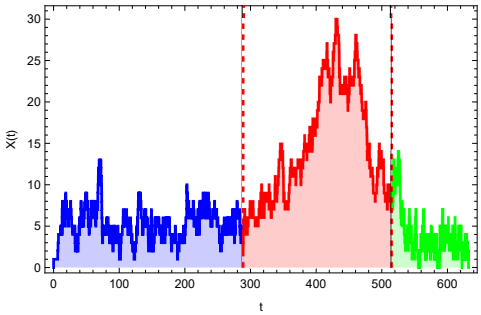

Figure 16. Time series of the number of customers in the $GI/G/10$ queueing system with real and estimated segments marked in blue, red and green with change points $t_1^* = 289$ and $t_2^* = 514$.

7. Reinforcement Learning for Dynamic Optimization

As is known, many dynamic optimization problems arising in queueing theory can be formulated as Markov (MDP), continuous-time Markov (CTMDP), or semi-Markov decision problems (SMDP) which can be solved using dynamic programming (DP) approach. In case of general distributed interarrival and service times, one defines the generalized semi-Markov decision problem (GSMDP). As shown in [53], arbitrary GSMDP can be approximated by MDP or CTMDP using PH distributions. In this case, instead of the original controlled queueing system $GI/G/K$, we obtain the equivalent queue of the type $PH/PH/K$. The classical DP algorithms, such as value iteration and policy iteration algorithms, generally face the problem of dimensionality. Moreover, these algorithms require knowledge of the transition probabilities and the immediate cost structure, which is not always explicitly available. Thus, an alternative approach should be used, for example, reinforcement learning (RL). RL is a type of ML technique in which an agent learns to make decisions by interacting with an environment. It can be treated as a simulation-based dynamic programming. Here we illustrate the application of the deep Q-learning

algorithm, where the Q-values are estimated using neural networks. This algorithm will be adapted to minimize the average cost per unit of time. For controlled queueing systems, the environment is a state space of the corresponding random process. In the dynamic control problem considered, it is assumed that all states of the system are observable. This assumption is used for a large number of controlled queueing systems. By presenting reinforcement learning in this paper, we wanted to show the principle of using this type of machine learning for a queueing system. However, although there are many publications on reinforcement learning, there is little description of its use in queueing theory. The presence of incomplete information on the process trajectory may degrade the estimation of the optimal policy but is not a critical problem since neural networks are used to estimate the quality function. This may require a larger number of episodes and associated computational time. The example we have chosen here to demonstrate the usage of the RL in the queueing theory can be easily generalized to a whole class of controlled queueing systems, where it is necessary to solve the problem of optimal resource allocation, scheduling, and routing.

7.1. DP Approach

For simplicity, let us take a concrete example of the CTMDP. Consider a queueing system $M/M/K$ with an arrival rate λ and a service rate μ. We chose a Markovian system because we can find an exact solution to the dynamic optimization problem, which we will then use to verify the results of reinforcement learning. However, in general, the RL approach can also be applied to the arbitrary system $GI/G/K$. The agent or controller can activate an idle server and deactivate a busy one. The server's activation and deactivation are associated with switching costs $c_s > 0$. The dynamics of the system are described by a continuous-time Markov chain $\{X(t)\}_{t\geq 0}$ with a state space E, where the random vector $X(t) = (Q(t), D(t))$ specifies the number of customers in the queue $Q(t) \in \mathbb{N}_0$ and in the service $D(t) \in \{0, 1, \ldots, K\}$. Define the action space $A = \{0, 1\}$. In case of a new arrival, $a = 0$ means keeping the idle server deactivated, and $a = 1$ means activating an idle server. In case of service completion, $a = 0$ means deactivating the released server and $a = 1$ means keeping the server activated. If there is no idle server, then at the moment of a new arrival, no control action must be performed. In this case, the customer joins the queue. If the queue is empty, then at service completion, the server becomes deactivated. Actions can be selected in states at the moment of arrival if there is an idle server in the system or at the moment of service completion if the queue is not empty. Define a control policy as a function $f = (f_0, f_1) : \tilde{E} \to A$ that for any state $x = (q, d) \in \tilde{E} \subset E$ from a subset of states $\tilde{E}$, where the control action can be performed, specifies the action $f_0(x)$ upon a new arrival in state x and the action $f_1(x)$ upon a service completion in state x. The optimization problem consists of the evaluation of optimal policy f^* which minimizes the long-run average cost

$$\begin{aligned} g^f = g^f(x) &= \lim_{t\to\infty} \frac{1}{t}\mathbb{E}^f\Big[\int_0^t c(X(u))du + c_s S(t)\Big|X(0) = x\Big] \\ &= \lim_{t\to\infty} \frac{1}{t}\mathbb{E}^f\Big[\int_0^t (q(X(u)) + d(X(u)))du + c_s S(t)\Big|X(0) = x\Big] \Rightarrow \min_f, \end{aligned} \tag{10}$$

where $c(X(u))$ is an immediate cost in state $X(u)$ associated with the number of customers in the queue and in service in this state, defined as $q(X(u)) + d(X(u))$, $S(t)$ is a random number of switches up to time t. The problem (10) was solved using the DP policy iteration algorithm. This algorithm iteratively improves the given arbitrary available policy by evaluating the Bellman equation for the state value function $V : E \to \mathbb{R}$, defined as

$$V(x) = \lim_{t\to\infty} \mathbb{E}^f\Big[\int_0^t (c(X(u)) - g^f)du + c_s S(t)\Big|X(0) = x\Big] \tag{11}$$

or, in detail for the system under study

$$V(x) = \frac{1}{\lambda + d(x)\mu}\Big[q(x) + d(x) - g^f + \lambda(V(x + e_{a_0}) + c_s 1_{\{a_0=1\}}) + d(x)\mu(V(x - e_0 - e_1 + e_{a_1} + c_s 1_{\{a_1=0\}})1_{\{q(x)>0\}}) + d(x)\mu(V(x - e_1) + c_s)1_{\{q(x)=0\}}\Big], \quad (12)$$

where e_j is a null vector with 1 at jth position starting from 0th, $q(x)$ and $d(x)$ are the components of a vector state x. The components of the optimal policy are defined as $f_0(x) = \arg\min\{V(x + e_0), V(x + e_1) + c_s\}$ and $f_1(x) = \arg\min\{V(x - e_1) + c_s, V(x - e_0)\}$. The system (12) is uniquely solved for the relative value function $v(x) = V(x) - V(x_0)$, where x_0 is a chosen reference state, for example $x_0 = (0, 0)$. The optimal policy for the proposed optimization problem (10) has a hysteresis form. This policy is defined by two threshold levels $a > b$ for the upper a and lower b bounds of the queue length to activate and deactivate the servers. For our system and given cost structure $b = 0$ for all states, $x \in E$ and a depends on the number of busy servers in state x. According to this policy, when a customer arrives in the state with d busy servers and the queue length becomes greater than $a(d)$, then another available idle server must be activated. Otherwise, the number of waiting customers increases by one. The activated servers remain active until they become empty in states with no waiting customers. The following are examples of specific calculations for the system $M/M/10$ with $\lambda = 10$ and $\mu = 7$:

- Case 1: $c_s = 0.4$, $g^* = 4.373$, $a^*(d) = \{2, 4, 7, 9, 12, 14, 17, 19, 22\}$ for $d = 1, 2, \ldots, 9$,
- Case 2: $c_s = 0.2$, $g^* = 3.507$, $a^*(d) = \{2, 3, 4, 6, 7, 9, 10, 12, 13\}$ for $d = 1, 2, \ldots, 9$.

The results were obtained for the queueing system with a truncated buffer. Here, the buffer capacity was set to 50, which guarantees a very small probability of loss of the customer. We use these results in the next subsection to check the quality of the RL algorithm.

7.2. RL Approach

Here, we briefly discuss such value-based RL methods as Q-learning (QL) and deep Q-learning (DQN). The latter algorithm will be used for the numerical experiments. Using an event-based simulator, we generate data sets D in the form of tuple sequences

$$D \leftarrow D \cup \{x, a_j, y, q(x) + d(x), \tau_{xy}, S(\tau_{xy})\}.$$

Here, x is a state of the system, a_j is an action for arrival if $j = 0$ and service completion if $j = 1$ associated with the state x, y is the next state after transition, $q(x) + d(x)$ is the number of customers in state x, and τ_{xy} is a holding time in state x before a transition to y, $S(\tau_{xy}) \in \{0, 1\}$ counts the number of activations/deactivations in a transition period. Moreover, the simulator calculates for the policy f the corresponding average cost g^f. To approximate the average cost g^f for the given policy f from (10), we have

$$\begin{aligned} g^f_{i+1} &= \frac{1}{t_{i+1}}\Big[\int_0^{t_{i+1}} c(X(u))du + c_s S(t_{i+1})\Big] = \frac{1}{t_{i+1}}\Big[\int_0^{t_i}(q(X(u)) + d(X(u)))du + (q(X(t_i)) + d(X(t_i)))(t_{i+1} - t_i) \\ &+ c_s S(t_i) + c_s(S(t_{i+1}) - S(t_i))\Big] \\ &= \frac{t_i}{t_{i+1}} g^f_i + \frac{t_{i+1} - t_i}{t_{i+1}}(q(X(t_i)) + d(X(t_i))) + c_s\frac{S(t_{i+1}) - S(t_i)}{t_{i+1}} \\ &= g^f_i + \frac{\tau_{xy}}{t_{i+1}}(q(x) + d(x) - g^f_i) + c_s\frac{S(\tau_{xy})}{t_{i+1}}, \end{aligned}$$

where $\{t_i\}_{i\in\mathbb{N}_0}$ is a sequence of transition moments from one state to another, $t_{i+1} - t_i = \tau_{xy}$ is a holding time in state x with $X(t_i) = x$, and $X(t_{i+1}) = y$. The transitions in the model

are based on residual times until the arrival of customers and until service completion at busy servers. Such a scheme of the simulation model allows for its use in systems with arbitrary time distributions.

The value-based algorithms, QL and DQN, calculate the action-state value function (quality function) $Q : E \times A \to \mathbb{R}$,

$$Q(x,a) = \lim_{t\to\infty} \mathbb{E}^f \Big[\int_0^t (q(X(u)) + d(X(u)) - g^f)du + c_s S(t) \Big| X(0) = x, a \Big] \tag{13}$$

which represents the expected utility (or quality) of taking action a in state $x \in E$ and following the optimal policy thereafter. The actions a_0 and a_1 are selected using an ε-greedy exploration-exploitation policy

$$a_0 = \begin{cases} \text{random action from } A = \{0,1\}, & \text{with probability } \varepsilon, \\ \arg\min\{Q(x,0), Q(x,1) + c_s\}, & \text{with probability } 1-\varepsilon \end{cases} \tag{14}$$

$$a_1 = \begin{cases} \text{random action from } A = \{0,1\}, & \text{with probability } \varepsilon, \\ \arg\min\{Q(x-e_1,0) + c_s, Q(x-e_0,1)\}, & \text{with probability } 1-\varepsilon. \end{cases}$$

The Q-value (13) update is then of the following form: the QL algorithm provides a Q-table update. If $X(0) = x$ and the selected action is a for the transition type j, then after a transition, the system state is $X(t_1) = y$ with the next selected action b and the type of transition k. In this case $\tau_{xy} = t_1$ specifies a holding time in state x before a transition to y. According to (13), we have

$$\begin{aligned} Q(x,a) = \mathbb{E}^f \Big[\int_0^{t_1} (q(X(u)) + d(X(u)) - g^f)du + c_s S(t_1) \Big| X(0) = x, a \Big] \\ + \lim_{t\to\infty} \mathbb{E}^f \Big[\int_{t_1}^t (q(X(u)) + d(X(u)) - g^f)du + c_s(S(t) - S(t_1)) \Big| X(t_1) = y, b, X(0) = x, a \Big]. \end{aligned} \tag{15}$$

Using the Robbins–Monro method of numerical evaluation of the expectation, where $\mathbb{E}[Z]$ can be evaluated from the sequence $\{z_l\}$ which converges to $\mathbb{E}[Z]$ with learning rate α_l,

$$z_{l+1} = z_l + \alpha_l(Z_l - z_l),$$

taking into account (15), the Q-values can be approximated by

$$\begin{aligned} Q(x,a_0) &\leftarrow Q(x,a_0) + \alpha[(q(x) + d(x) - g^f)\tau_{xy} + c_s 1_{\{a_0=1\}} \\ &\quad + Q(y, \arg\min\{Q'(y,0), Q'(y,1) + c_s\}) - Q(x,a_0)],\ b = b_0, \\ Q(x,a_1) &\leftarrow Q(x,a_1) + \alpha[(q(x) + d(x) - g^f)\tau_{xy} + c_s 1_{\{a_1=0\}} \\ &\quad + Q(y, \arg\min\{Q'(y,0), Q'(y,1) + c_s\}) - Q(x,a_1)],\ b = b_0, \\ Q(x,a_0) &\leftarrow Q(x,a_0) + \alpha[(q(x) + d(x) - g^f)\tau_{xy} + c_s 1_{\{a_0=1\}} \\ &\quad + Q(y, \arg\min\{Q'(y-e_1,0) + c_s, Q'(y-e_0,1)\}) - Q(x,a_0)],\ b = b_1, \\ Q(x,a_1) &\leftarrow Q(x,a_1) + \alpha[(q(x) + d(x) - g^f)\tau_{xy} + c_s 1_{\{a_1=0\}} \\ &\quad + Q(y, \arg\min\{Q'(y-e_1,0) + c_s, Q'(y-e_0,1)\}) - Q(x,a_1)],\ b = b_1, \end{aligned} \tag{16}$$

for different transition types in state x and y, where α is a learning rate and Q' is the old vector of Q-values which define new policy to be evaluated. Next, we present the main steps of the algorithm.

1. **Initialization.** Initialize quality function $Q(x,a) = 0, x \in E, a \in A$, $Q'(x,a) = Q(x,a), I(x) = 0, x \in E$ which counts the times of occurrence of the state x.

2. **Policy evaluation.** While Q-values are not converged, perform the following steps.
3. **Evaluation and update of the sample** Q. $D \leftarrow$ simulate(). For

$$\{x, a_j, y, q(x) + d(x), \tau_{xy}, S(\tau_{xy})\} \in D$$

update the values of Q with respect to (16) with a learning rate $\alpha = \frac{1}{I(x)+1}$.
4. **Policy improvement.** Q to Q' conversion. After a convergence in one episode, a new policy is generated using the relation $Q'(x,a) \leftarrow Q(x,a)$. Then set

$$f_0(x) = \arg\min\{Q'(x,0), Q'(x,1) + c_s\},$$
$$f_1(x) = \arg\min\{Q'(x - e_1, 0) + c_s, Q'(x - e_0, 1)\}$$

for all $x \in \tilde{E}$.

The value ε in (14) is initialized to $\varepsilon = 1$ and updated with decay according to the rule $\varepsilon = 1 - (k-1)0.001$, where k is an episode index. In this case, if $k = 1$, then $\varepsilon = 0$, and the actions should be chosen at random. If k is equal to the maximum number of episodes $K = 1000$, then $\varepsilon = 0$ and the decision is made based on the estimated Q-values.

If the state space is large and the queueing system operates with low and medium load factors, then there is a situation where Q values remain equal to zero for a large number of pairs (x, a), which makes a lookup table for each state action pair infeasible. A possible solution is to approximate the values of Q using neural networks. Deep Q-learning updates Q-values by minimizing the loss between predicted and target Q-values using gradient descent. The main steps of the DQN algorithm are listed below.

1. **Initialization.** Initialize the policy network $Q_W(x,a)$ with random weights W, initialize D. For $k = 1$ to K, perform the following steps.
2. $D \leftarrow simulate()$.
3. **Calculate the target** Q**-value**

$$\hat{Q} = (q(x) + d(x) - g^f)\tau_{xy} + c_s 1_{\{a_0=1\}} + Q_W(y, \arg\min\{Q_W(y,0), Q_W(y,1) + c_s\}),\ b = b_0,$$
$$\hat{Q} = (q(x) + d(x) - g^f)\tau_{xy} + c_s 1_{\{a_1=1\}} + Q_W(y, \arg\min\{Q_W(y,0), Q_W(y,1) + c_s\}),\ b = b_0,$$
$$\hat{Q} = (q(x) + d(x) - g^f)\tau_{xy} + c_s 1_{\{a_0=1\}} + Q_W(y, \arg\min\{Q_W(y - e_1, 0) + c_s, Q_W(y - e_0, 1)\}),\ b = b_1,$$
$$\hat{Q} = (q(x) + d(x) - g^f)\tau_{xy} + c_s 1_{\{a_1=1\}} + Q_W(y, \arg\min\{Q_W(y - e_1, 0) + c_s, Q_W(y - e_0, 1)\}),\ b = b_1.$$

4. **Calculate the loss** between predicted and target Q-values

$$L(W) \leftarrow \frac{1}{|D|} \sum_{\{x, a_j, y, q(x)+d(x), \tau_{xy}, S(\tau_{xy})\} \in D} [\hat{Q} - Q_W(x,a)]^2. \tag{17}$$

5. **Back propagate the loss** and update the weights $W \leftarrow W - \alpha \nabla_W L(W)$.
6. **Periodically update** $W' \leftarrow W$.

Figure 17 illustrates the architectures of the Q-network and the corresponding training network that is used to update the Q-network with respect to the loss of the mean square (MS) (17). For the system state, we use a feature vector

$$x' = \frac{x}{N} \tag{18}$$

normalized by N to ensure that $|x'| < 1$. As we can see, the Q-network is a neural network that takes a state x as input (for 5 servers the length of the state vector is 6) and outputs a Q-value $Q(x,a)$ for each possible action a. Here, two fully connected linear layers with the output vectors of sizes 400 and 300 and an activation function ReLU are used. A separate

training network is introduced to optimize the MS between the Q-network and the target Q-value $\hat{Q}$.

Figure 17. The architectures of Q-network and training network to update the Q network.

In Figure 18, we present the convergence rates for the average cost of the DQN algorithm for the parameter of Cases 1 and 2 proposed in the previous subsection. The dashed red line illustrates the actual optimal value obtained by the DP approach. The data sets obtained were smoothed a bit so that, on the one hand, it was possible to track the convergence of the algorithm and, on the other hand, to preserve the fluctuations inherent in this approach. In addition to calculating neural network parameters that minimize the average cost per unit of time, reinforcement learning can also be used to test convergence to the optimal solution for different types of distribution. Hence, it is possible to analyze the sensitivity of the optimal control policy to the form of distribution in complex controlled queueing systems.

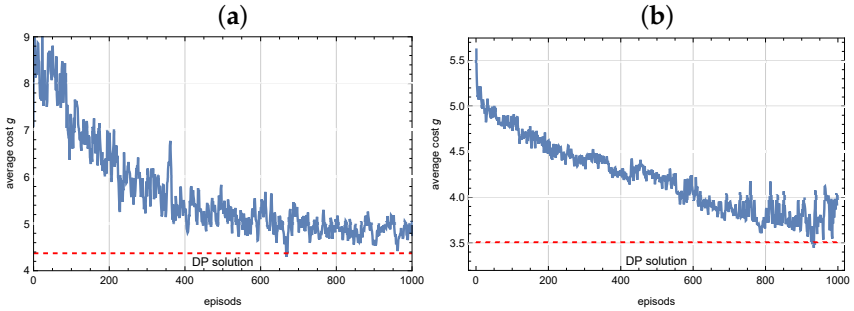

Figure 18. Convergence rates of the DQN for Case 1 (**a**) and Case 2 (**b**).

We can see that applying RL to queueing systems can offer significant advantages but also presents notable challenges. RL can adapt to changes in the queueing system's environment, enabling real-time decision-making for tasks like load balancing, scheduling, and resource allocation, and does not require explicit system modeling. Advanced RL algorithms such as DQN can scale to large queueing systems or distributed networks. However, it should be noted that training RL models requires significant computational resources, especially in complex queueing systems with high-dimensional state and action spaces. RL algorithms may converge slowly or become unstable in this case.

8. Conclusions

We have presented typical problems in queueing theory that arise in the framework of performance analysis and optimization for which ML algorithms can be applied. As an example of a system, we have taken the $GI/G/K$ model and solved various regression, classification, and optimization problems. We believe that this approach is quite universal and suitable for arbitrary systems. Of course, a number of preparatory works are necessary for any arbitrary queueing system. A high-quality simulator is needed. It can be tested with the analytical results available from the simplified system. Next, it is necessary to

empirically test the quality of the approximating parametric function and then to create samples of the necessary volume for further training and testing of the ML algorithms used. The results of reinforcement learning were verified using the equivalent dynamic programming approach. It is obvious that the application of machine learning in queueing theory has definitive advantages but also certain difficulties in realization. However, the development of artificial intelligence methods and their synergy with queueing systems significantly expands the possibilities of queueing theory and is an effective addition to the existing classical methodology.

Author Contributions: Conceptualization, D.E. and V.V.; methodology, D.E., V.V. and J.S.; software, N.S.; validation, N.S.; formal analysis, D.E. and N.S.; investigation, D.E. and N.S.; resources, J.S.; writing— original draft, D.E.; writing—review & editing, N.S. All authors have read and agreed to the published version of the manuscript.

Funding: The research was funded by the Austro-Hungarian Cooperation (OMAA) Grant No. 116öu7.

Data Availability Statement: The study did not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Oladipupo, T. Types of Machine Learning Algorithms. In *New Advances in Machine Learning*; InTech: Rijeka, Croatia, 2010; Chapter 3. [CrossRef]
2. Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning*; Springer: New York, NY, USA, 2009. [CrossRef]
3. Bishop, C.M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*; Springer: Berlin/Heidelberg, Germany, 2006.
4. Sutton, R.; Barto, A. Reinforcement Learning: An Introduction. *IEEE Trans. Neural Netw.* **1998**, *9*, 1054. [CrossRef]
5. Choi, D.; Kim, N.; Chae, K. A Two-Moment Approximation for the GI/G/c Queue with Finite Capacity. *INFORMS J. Comput.* **2005**, *17*, 75–81. [CrossRef]
6. Shore, H. Simple Approximations for the GI/G/c Queue-I: The Steady-State Probabilities. *J. Oper. Res. Soc.* **1988**, *39*, 279. [CrossRef]
7. Shore, H. Simple Approximations for the GI/G/c Queue-II: The Moments, the Inverse Distribution Function and the Loss Function of the Number in the System and of the Queue Delay. *J. Oper. Res. Soc.* **1988**, *39*, 381–391. [CrossRef]
8. Shortle, J.; Thompson, J.; Gross, D.; Harris, C. *Fundamentals of Queueing Theory*; Wiley: Hoboken, NJ, USA, 2018. [CrossRef]
9. Van Hoorn, M.; Seelen, L. Approximations for the GI/G/c queue. *J. Appl. Probab.* **1986**, *23*, 484–494. [CrossRef]
10. Whitt, W. The Queueing Network Analyzer. *Bell Syst. Tech. J.* **1983**, *62*, 2779–2815. [CrossRef]
11. Shin, Y.; Moon, D. On approximations for GI/G/c retrial queues. *J. Appl. Math. Inform.* **2013**, *31*, 311–325. [CrossRef]
12. Telek, M.; Heindl, A. Matching Moments For Acyclic Discrete And Continuous Phase-Type Distributions Of Second Order. *Int. J. Simul. Syst. Sci. Technol.* **2003**, *3*, 47–57.
13. Vishnevsky, V.; Gorbunova, A.V. Application of Machine Learning Methods to Solving Problems of Queuing Theory. In *Information Technologies and Mathematical Modelling. Queueing Theory and Applications*; Springer International Publishing: Cham, Switzerland, 2022; pp. 304–316. [CrossRef]
14. Stintzing, J.; Norrman, F. Prediction of queuing behaviour through the use of artificial neural networks. In Proceedings of the Computer Science. 2017. Available online: https://api.semanticscholar.org/CorpusID:198923796 (accessed on 3 February 2025).
15. Nii, S.; Okudal, T.; Wakita, T. A Performance Evaluation of Queueing Systems by Machine Learning. In Proceedings of the 2020 IEEE International Conference on Consumer Electronics—Taiwan (ICCE-Taiwan), Taoyuan, Taiwan, 28–30 September 2020; IEEE: Piscataway, NJ, USA, 2020. [CrossRef]
16. Sherzer, E.; Senderovich, A.; Baron, O.; Krass, D. Can machines solve general queueing systems? *arXiv* **2022**. [CrossRef]
17. Kyritsis, A.I.; Deriaz, M. A Machine Learning Approach to Waiting Time Prediction in Queueing Scenarios. In Proceedings of the 2019 Second International Conference on Artificial Intelligence for Industries (AI4I), Laguna Hills, CA, USA, 25–27 September 2019; IEEE: Piscataway, NJ, USA, 2019. [CrossRef]
18. Sivakami, S.; Senthil, K.; Yamini, S.; Palaniammal, S. Artificial neural network simulation for markovian queueing models. *Indian J. Comput. Sci. Eng.* **2020**, *11*, 127–134. [CrossRef]

19. Hijry, H.; Olawoyin, R. Predicting Patient Waiting Time in the Queue System Using Deep Learning Algorithms in the Emergency Room. *Int. J. Ind. Eng. Oper. Manag.* **2021**, *3*, 33–45. [CrossRef]
20. Chocron, E.; Cohen, I.; Feigin, P. Delay Prediction for Managing Multiclass Service Systems: An Investigation of Queueing Theory and Machine Learning Approaches. *IEEE Trans. Eng. Manag.* **2022**, *71*, 4469–4479. [CrossRef]
21. Dieleman, N.; Berkhout, J.; Heidergott, B. A neural network approach to performance analysis of tandem lines: The value of analytical knowledge. *Comput. Oper. Res.* **2023**, *152*, 106124. [CrossRef]
22. Vishnevsky, V.; Klimenok, V.; Sokolov, A.; Larionov, A. Performance Evaluation of the Priority Multi-Server System MMAP/PH/M/N Using Machine Learning Methods. *Mathematics* **2021**, *9*, 3236. [CrossRef]
23. Klimenok, V.; Dudin, A.; Vishnevsky, V. Priority multi-server queueing system with heterogeneous customers. *Mathematics* **2020**, *8*, 1501. [CrossRef]
24. Vishnevsky, V.; Larionov, A.; Mukhtarov, A.; Sokolov, A. Investigation of Tandem Queuing Systems Using Machine Learning Methods. *Control Probl.* **2024**, *4*, 13–25. [CrossRef]
25. Vishnevsky, V.; Larionov, A.; Semenova, O.; Ivanov, R. State Reduction in Analysis of a Tandem Queueing System with Correlated Arrivals. In *Information Technologies and Mathematical Modelling. Queueing Theory and Applications*; Dudin, A., Nazarov, A., Kirpichnikov, A., Eds.; Springer: Cham, Switzerland, 2017; pp. 215–230.
26. Klimenok, V.; Dudina, O.; Vishnevsky, V.; Samouylov, K. Retrial Tandem Queue with BMAP-Input and Semi-Markovian Service Process. In *Distributed Computer and Communication Networks*; Vishnevskiy, V., Samouylov, K., Kozyrev, D., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 159–173.
27. Vishnevsky, V.; Semenova, O.; Bui, D. Using a Machine Learning Approach for Analysis of Polling Systems with Correlated Arrivals. In *Distributed Computer and Communication Networks: Control, Computation, Communications*; Springer International Publishing: Cham, Switzerland, 2021; pp. 336–345. [CrossRef]
28. Vishnevsky, V.; Semenova, O. Polling Systems and Their Application to Telecommunication Networks. *Mathematics* **2021**, *9*, 117. [CrossRef]
29. Efrosinin, D.; Vishnevsky, V.; Stepanova, N. A Machine-Learning Approach To Queue Length Estimation Using Tagged Customers Emission. In *Distributed Computer and Communication Networks: Control, Computation, Communications 2023*; Springer International Publishing: Cham, Switzerland, 2024; pp. 265–276. [CrossRef]
30. Vishnevsky, V.; Klimenok, V.; Sokolov, A.; Larionov, A. Investigation of the Fork–Join System with Markovian Arrival Process Arrivals and Phase-Type Service Time Distribution Using Machine Learning Methods. *Mathematics* **2024**, *12*, 659. [CrossRef]
31. Ivanova, N.; Vishnevsky, V. Application of k-out-of-n:G System and Machine Learning Techniques on Reliability Analysis of Tethered Unmanned Aerial Vehicle. In *Proceedings of the Information Technologies and Mathematical Modelling. Queueing Theory and Applications*; Dudin, A., Nazarov, A., Moiseev, A., Eds.; Springer: Cham, Switzerland, 2022; Volume 1605, pp. 117–130.
32. Alfa, A.S.; Abu Ghazaleh, H. Machine Learning Tool for Analyzing Finite Buffer Queueing Systems. *Mathematics* **2025**, *13*, 346. [CrossRef]
33. Matsumoto, Y. On optimization of polling policy represented by neural network. *ACM SIGCOMM Comput. Commun. Rev.* **1994**, *24*, 181–190. [CrossRef]
34. Kohonen, T. The self-organizing map. *Proc. IEEE* **1990**, *78*, 1464–1480. [CrossRef]
35. Efrosinin, D.; Rykov, V.; Stepanova, N. Evaluation and Prediction of an Optimal Control in a Processor Sharing Queueing System with Heterogeneous Servers. In *Distributed Computer and Communication Networks*; Springer International Publishing: Cham, Switzerland, 2020; pp. 450–462. [CrossRef]
36. Efrosinin, D.; Vishnevsky, V.; Stepanova, N. Optimal Scheduling in General Multi-Queue System by Combining Simulation and Neural Network Techniques. *Sensors* **2023**, *23*, 5479. [CrossRef]
37. Li, Q.L.; Ma, J.Y.; Fan, R.N.; Xia, L. An Overview for Markov Decision Processes in Queues and Networks. *arXiv* **2019**. [CrossRef]
38. Özkan, E.; Kharoufeh, J.P. Optimal control of a two-server queueing system with failures. *Probab. Eng. Inf. Sci.* **2014**, *28*, 489–527. [CrossRef]
39. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
40. Li, Y. Deep Reinforcement Learning: An Overview. *arXiv* **2017**. [CrossRef]
41. Van Hasselt, H.; Guez, A.; Silver, D. Deep Reinforcement Learning with Double Q-Learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30. [CrossRef]
42. Thomas, P.S.; Brunskill, E. Policy Gradient Methods for Reinforcement Learning with Function Approximation and Action-Dependent Baselines. *arXiv* **2017**. [CrossRef]
43. Konda, V.; Tsitsiklis, J. Actor-Critic Algorithms. In *Proceedings of the Advances in Neural Information Processing Systems*; Solla, S., Leen, T., Müller, K., Eds.; MIT Press: Cambridge, MA, USA, 1999; Volume 12.
44. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**. [CrossRef]

45. Okumu, E.M. A deep reinforcement learning approach to queue management and revenue maximization in multi-tier 5G wireless networks. *Am. Acad. Sci. Res. J. Eng. Technol. Sci. (ASRJETS)* **2021**, *84*, 15–26.
46. Jali, N.; Qu, G.; Wang, W.; Joshi, G. Efficient Reinforcement Learning for Routing Jobs in Heterogeneous Queueing Systems. *arXiv* **2024**. [CrossRef]
47. Efrosinin, D.; Vishnevsky, V.; Stepanova, N. Simulation-Based Optimization for Resource Allocation Problem in Finite-Source Queue with Heterogeneous Repair Facility. In *Distributed Computer and Communication Networks*; Vishnevsky, V.M., Samouylov, K.E., Kozyrev, D.V., Eds.; Springer: Cham, Switzerland, 2025; pp. 187–202.
48. Rebuffi, L.S. Reinforcement Learning Algorithms for Controlled Queueing Systems. Ph.D. Thesis, University of Grenoble Alpes, Grenoble, France, 2014.
49. Asmussen, S. *Applied Probability and Queues*; Springer: New York, NY, USA, 2003. [CrossRef]
50. Bolch, G.; Greiner, S.; de Meer, H.; Trivedi, K.S. *Steady-State Solutions of Markov Chains*; Wiley: Hoboken, NJ, USA, 1998. [CrossRef]
51. Cox, D. A use of complex probabilities in the theory of stochastic processes. *Math. Proc. Camb. Philos. Soc.* **1955**, *51*, 313–319. [CrossRef]
52. Choudhury, A. A Simple Derivation of Moments of the Exponentiated Weibull Distribution. *Metrika* **2005**, *62*, 17–22. [CrossRef]
53. Younes, H.L.S.; Simmons, R.G. Solving Generalized Semi-Markov Decision Processes Using Continuous Phase-Type Distributions. In Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, San Jose, CA, USA, 25–29 July 2004; McGuinness, D.L., Ferguson, G., Eds.; AAAI Press/The MIT Press: Cambridge, MA, USA, 2004; pp. 742–748.