Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

CAN MACHINES SOLVE GENERAL QUEUEING PROBLEMS?

Opher Baron
Dmitry Krass
Eliran Sherzer

Rotman School of Management
University of Toronto
105 St George St
Toronto, M5S 3E6, CANADA

Arik Senderovich

School of Information Technology
York University
4700 Keele St
Toronto, M3J 1P3, CANADA

ABSTRACT

We study how well a machine can solve a general problem in queueing theory, using a *neural net* to predict the stationary queue-length distribution of an $M/G/1$ queue. This problem is, arguably, the most general queuing problem for which an analytical "ground truth" solution exists. We overcome two key challenges: (1) generating training data that provide "diverse" service time distributions, and (2) providing continuous service distributions as input to the neural net. To overcome (1), we develop an algorithm to sample phase-type service time distributions that cover a broad space of non-negative distributions; exact solutions of $M/PH/1$ (with phase-type service) are used for the training data. For (2) we find that using only the first n moments of the service times as inputs is sufficient to train the neural net. Our empirical results indicate that neural nets can estimate the stationary behavior of the $M/G/1$ extremely accurately.

1 INTRODUCTION

In this paper we attempt to "teach" a machine to compute performance measures for a general queueing system. In effect, we ask whether a machine can replace complicated mathematical methods and provide fast and accurate solutions for general problems in queueing theory. We show that the answer is likely in the *affirmative*.

The specific problem we select is the computation of the stationary distribution of an $M/G/1$ queue (Poisson arrivals, general service, single server). We focus on this problem, because it lies "on the cusp" of the analytical frontier: while exact solutions exist, they are both computationally and mathematically complex. On the other hand, the problem (specifically the service time distribution) is general.

Our study is motivated by major successes of machine learning algorithms and, more specifically, neural models, in a variety of fields ranging from computer vision, to natural language processing, to traffic prediction in transportation systems, see for example (LeCun et al. 2015). Within the Operations Research (OR) domain, recent papers have shown the usefulness of neural networks in well-known problems such as scheduling, resource allocation, and others, see (Xiao and Bhardwaj 2018).

Typically, queueing systems are complex, hard to analyze, and analytical methods often struggle even with simple stylized models. Furthermore, real-life queueing systems introduce complexities that often make models analytically intractable, e.g., non-stationarity, customer abandonment, correlations between arrivals and service times, and many more. In many cases, since no analytic solution is available, such systems can only be analyzed via simulation; typically using the traditional discrete time approach. Yet, traditional simulation models have their own drawbacks, ranging from the requirement for a detailed specification of all underlying distributions to, often, a long convergence time.

Although more advanced techniques, such as Lindley's recursion, e.g., in (Palomo and Pender 2020) and (Vazquez-Avila and Sandoval-Arechiga 2019), allow queueing models such as the $G/G/1$ queue to

be simulated accurately within seconds, this is only true if the arrival and service time distributions are "easy" to sample from; in real-life systems, those distributions may be complex. Furthermore, Lindley's recursion only provides a sequence of waiting times. Converting these into steady-state probabilities of the queue length, which is required in order to provide additional performance measures, is a computational process which can result in an accuracy loss. In contrast, the technique we develop can easily handle *any* service time distribution and provides accurate results instantaneously.

Applying a neural model to an $M/G/1$ queue raises two major challenges. First, generating a sufficiently diverse set of training instances, i.e., pairs of arrival rates and service time distributions, that can well represent any general positive-valued distribution. A second, and related, challenge is "feeding" continuous service time distributions as an input to the neural network. Clearly, a general continuous distribution cannot be used directly; thus, we must represent it by a tensor with, possibly, some information loss. Thus, we must develop an appropriate distribution-to-tensor mapping.

To deal with the first challenge, we generate our training set using Phase-Type (PH) family of distributions. This family is known to be dense within the class of all non-negative distribution functions, i.e., a PH distribution with a sufficient number of phases can, theoretically, approximate any non-negative distribution to an arbitrary precision, see (Asmussen 2003a Theorem 4.2). Moreover, a $M/PH/1$ system (PH-type service time distribution) allows for a computationally efficient solution methodology compared to a general $M/G/1$ queue. Sampling from the universe of PH distributions is not straightforward, as its parameters are not independent and have multiple constraints that must be enforced. To accommodate, we introduce a versatile sampling method that is shown to generate a diverse set of distributions.

To address the second challenge, we represent the service times using the first n moments of the distribution. These moments can be easily computed from empirically observed service times. The number of moments n is a hyper-parameter, which is tuned as part of the training process. Representing service time distributions using a finite number of moments is natural in view of existing queueing results that often depend only on the first two moments of the service and inter-arrival time distribution, such as, the Khintchine–Pollaczek formula (Asmussen 2003b) for the mean queue length for the $M/G/1$. While it is clear that the moments of the distribution play a key role in queueing, we also know that different distributions can have identical first n moments. Thus, we do not know *a priori* how many moments are actually needed to have an accurate representation of the entire queue length distribution.

Our empirical results show that a neural network can approximate the stationary queue-length distribution for $M/G/1$ very closely, while using a relatively small net with roughly $275,000$ parameters. Further, we show that the only the first $n = 5$ moments of service time suffice to determine the distribution of the $M/G/1$ stationary queue length to very high level of accuracy; there is no substantial gain in accuracy when adding higher-order moments. This result is very intriguing as it provides further understanding of the dynamics of an $M/G/1$ queue and opens the window to analyze more complex systems. This result simplifies the task of fitting a particular parametric distribution to observed service time, as only the first few moments have to be matched. This is in contrast to methods such as the Expectation Maximization (EM) algorithm (Telek and Horváth 2007) that are much more time consuming. The main contributions of this paper are twofold: (i) We show that a data-driven machine learning technique based on neural networks effectively and efficiently solves a fundamental queueing problem while handling service time distributions of any complexity. (ii) We introduce an algorithm for sampling a "general" distribution, leveraging the versatility of the phase-type family of distributions and demonstrate its use in generating training samples; this algorithm has applicability far beyond the particular queueing model analyzed in the current paper.

Section 2 provides an end-to-end overview of our solution and details related literature. We present our PH sampling technique in Section 3, and describe our neural architecture, input pre-processing, and loss function in Section 4. The main findings from our experiments are provided in Section 5, followed by an evaluation of the approach on empirical data in Section 6. Section 7 provides concluding remarks and several directions for future work.

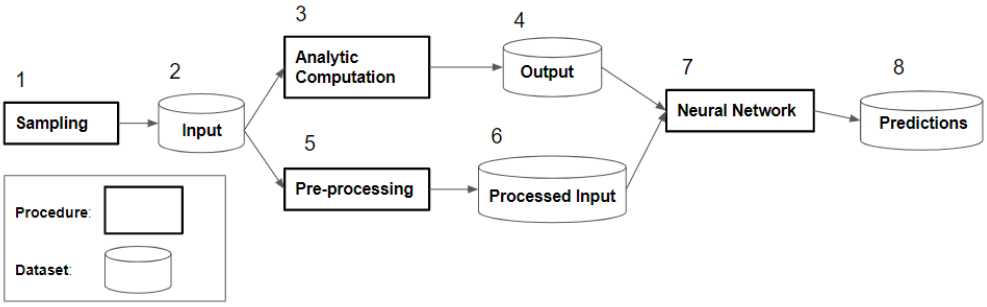

Figure 1: Work-flow diagram of our learning procedure.

2 SOLUTION OVERVIEW AND RELATED WORK

The main phases of our solution are presented in Figure 1. Steps 1-4 represent the generation of training input and output data, as detailed in Section 3. In step 1, "sampling", we generate input samples, as detailed in Section 3.1. The sample is generated in pairs, with each pair comprising an arrival rate and a service distribution (we only consider stable pairs, i.e., the arrival rate must be smaller than the service rate). This results in the "Training Input" data set in (step 2). In step 3, "Analytic Computation", we compute the stationary queue-length distribution using a well-known Quasi Birth and Death (QBD) method, as detailed in Section 3.2). This results in the output (step 4) that is then used as the target when learning from the training data. In step 5, "Pre-processing," we first compute the first n moments of the service time distribution, where the hyper-parameter n requires tuning, and then standardize the computed moments, as detailed in Section 4.1. The processed input and training output are fed into the neural network for training (step 7), as detailed in Section 4.2. Finally, in step 8, the model outputs the predicted stationary queue-length distribution in the same format as the training output.

2.1 Related Work

There have been only a few prior studies that use neural networks to analyze stationary behaviour in queues. The most similar paper to ours is by (Nii et al. 2020); it proposes a neural model to analyze a $GI/G/s$ queue. The input includes only the first two moments of the inter-arrival and service times distribution. Moreover, they only evaluated the average waiting time and their results were not very accurate. (Kyritsis and Deriaz 2019) also used a neural network to predict waiting times when focusing on an online prediction conditioning on the network current state. In (Pender and Zhang 2021) the authors study the possibility of accurately predicting the response times in real time of the G/G/1 processor sharing queue using several ML methods, including neural networks. The two main differences is that they computed the mean response time and not stationary behavior of the system and the inter-arrival and service time are sampled from exponential and log-normal distributions, as opposed to PH which covers a much larger state-space.

Two papers sample general PH distributions for given number of states. The procedure in (Okamura and Dohi 2015) samples the exact same PH distribution repeatedly (i.e., the same initial probability vector and the transition matrix) and thus cannot provide a diverse set of distributions. While the package offered in (Horváth and Telek 2019) sample different PH distributions; we show (in Section 3.1.4) that their sampling method covers a fairly narrow range of distributions, which is insufficient for our purposes.

3 DATA GENERATION

In this section we outline the input sampling procedure (Section 3.1), and the training output generation step (Section 3.2). To ensure stability of the sampled queues we first sample the service time distribution and then the arrival rate over the range limited from above by the sampled service rate. The main challenge

here in the sampling is to sample service distributions that provide a "wide" coverage of the state-space of all positive distributions. To this end, we employ PH distributions.

3.1 Sampling Phase-Type Distributions

A PH distribution can be represented by a random variable describing the time until absorption of a continuous-time Markov chain (CTMC) with a single absorbing state; all other states are transient. Each of the transient states of the CTMC corresponds to one of the states of the PH distribution, where a state sojourn time follows an exponential distribution. To fully represent a PH distribution with $m \geq 1$ states, we specify two parameters, namely α, the initial probability m-vector of starting the process in any of the m states, and S, the corresponding generating $m \times m$ matrix of the CTMC. In order for the pair (α, S) to be valid, it needs to meet the following constraints: (1) $S[i,i] < 0, \forall i \leq m$, (2) $\sum_{j=1}^{m} S[i,j] \leq 0$, and, (3) $\sum_{i=1}^{m} \alpha[i] = 1$, where $\alpha[i] \geq 0, \forall i \leq m$.

While, theoretically, a PH distribution can approximate any positively-valued distribution, we have no guarantees that a particular PH generation procedure would lead to particularly diverse set of distributions. We hypothesize that diversity is related to the underlying structure of the CTMC. In particular, random generation of S that satisfies the constraints above (the procedure followed in (Horváth and Telek 2019)), results in a single set of interconnected states and, in our experience, typically leads to poor diversity (this is confirmed by our computational results). Thus we develop a more sophisticated generating procedure aimed at having a variety of transition structures.

3.1.1 Challenges in Sampling: Diversifying the Transition Structure of the PH

For a general input, we wish to control two degrees of freedom hidden in α and S: (1) the size (number of states) of the PH distribution, and (2) the transition function that governs the switches between the different states. The former is relatively straightforward to deal with: we uniformly sample a value between 1 and max_{ph}, the maximum PH size that we use for our experiments (a hyper-parameter that be tuned by the user). The second consideration is not trivial and requires some effort.

When generating α and S, we wish to allow different state-to-state transition structures that are determined by three elements: (i) the initial probability vector α, (ii) the locations of the non-diagonal cells in S in which the value is 0 ($S[i,j] = 0$ implies direct transition from i to j is not possible), and (iii) the sum of each row in S. Clearly, if all non-diagonal cells in row i are 0, then the underlying CTMC will reach its absorption state once the system exits state i. We refer to such states as *fully-absorbing*. These are controlled by element (ii). Letting s_i be the sum of row i in S, i.e., $\sum_{j=1}^{m} S[i,j] = s_i$, if $s_i < 0$, then with probability $s_i/S[i,i]$ the CTMC will reach its absorption state after exiting state i. We refer to such states as *partially absorbing*. However, if $s_i = 0$, then the CTMC cannot reach the absorption state once the system exits state i. We refer to such states as *non-absorbing*. The composition of partially absorbing and non-absorbing states is controlled by element (iii). Thus, in order to control the structure of the PH distribution we need to control for elements (i), (ii), and (iii).

3.1.2 Classes of States in PH

We define a *class* as a set of states such that if we reach one of them from the initial state, it is impossible to reach states outside this set before the underlying CTMC reaches its absorption state. Each class, in fact, is a PH distribution on its own. Thus, we can sample each class separately and then group them together for a singe PH representation. We first sample the number of classes and their sizes, and only then sample the specific states within each class. This allows us to better control the sampled PH and create numerous peak points in the probability distribution function (PDF) of the sampled service time distributions.

The total number of classes may vary from 1 to m as it cannot exceed the total number of states. We note that the PH random variable has, by definition, a finite expected value. In order to avoid infinite expected values, each class must adhere to two conditions: (1) having at least a single partially or fully absorbing

Algorithm 1 Sample PH of a single class.

INPUT: $Size_i$
OUTPUT: α^i, S^i

procedure SAMPLING CLASS $i(Size_i)$
  $\alpha^i \leftarrow Zeros(1,Size_i), S^i \leftarrow Zeros(Size_i,Size_i)$, $Trans \leftarrow Zeros(Size_i,Size_i)$
  $FullA_i, PartA_i, NonA_i \leftarrow$ Sampling state types randomly $(Size_i)$ ▷ Assigning to different types
  $Trans \leftarrow$ Sample Trans$(Trans, Size_i, PartA_i, NonA_i)$
  $S^i[j,j] \leftarrow -U(1,1000), \forall j\{j \leq Size_i\}$
  $S^i[j,k] \leftarrow U(1,1000), \forall j,k\{j \neq k, Trans[j,k] = 1\}$
  $p_j \leftarrow U(0,1)$, $\forall j \in PartA_i$ ▷ p_j is the absorbing probability of state j
  $S^i[j,k] = S^i[j,k](1-p_j)\frac{-S^i[j,j]}{\sum_{l=1,l\neq j}^{Size_i} S^i[j,l]}$,$\forall j \in PartA_i, \forall k \neq j$
  $S^i[j,k] = S^i[j,k]\frac{-S^i[j,j]}{\sum_{l=1,j\neq j}^{Size_i} S^i[j,l]}$, $\forall j \notin PartA_i \forall, k \neq j$
  $\alpha^i \leftarrow Dirichlet(a_1,\ldots,a_{Size_i})$, $a_j \sim U(0,1), \forall j \leq Size_i$
  Return α^i, S^i
end procedure
procedure SAMPLE TRANS$(Trans, Size_i, PartA_i, NonA_i)$
  $NumTrans_j \leftarrow 1 + Binomial(Size_i - 2, U(0,1))$, $\forall j \in PartA_i, NonA_i$
  $Seleced_j \leftarrow Choose(\{k \leq Size_i, k \neq j\}, NumTrans_j)$, $\forall j \leq Size_j$
  $Trans[j,k] = 1$, $\forall j \leq Size_i, k \in Seleced_j$
  Return $Trans$
end procedure

state, and (2) any non-absorbing state must have a positive probability to reach one of the partially or fully absorbing states. Otherwise, the system may enter a "livelock", i.e., a loop in which it never reaches the absorbing state. We enforce (1) directly in our sampling methodology, while (2) is enforced by checking whether the expected value of the generated distribution is finite. If the value is infinite, we reject the sample (such rejections occurred in our experiments approximately 5% of the time).

3.1.3 Sampling Algorithms

We next briefly describe our sampling procedure; a more detailed description can be found in the working paper in (Sherzer et al. 2022). The main idea involves partitioning the PH into classes, sample each class independently, and then aggregate all classes into a unified PH representation (α, S). We first sample the number of states m (choosing $m = 1000$; this hyper-parameter can be set by the user) of the PH, the number of classes ($NumClasses$) and the initial probability of each class (P). Once these are generated, we sample the PH representation of class i $\forall i \leq NumClasses$, starting by generating the size of the class $Size_i$.

We note that sampling $Size_i$, should be done carefully, as the constraint $\sum_{i=1}^{NumClasses} Size_i = m$ must be met. To this end, $Size_i$ is sampled uniformly between 1 and $m - AssignedStates - (NumClasses - i)$. The upper-bound ensures that we assign only from the remaining states, (those that are not assigned by the previous classes: $1,\ldots,i-1$), and that there will be at least one state for all future classes (states $i+1,\ldots,NumClasses$).

Next, we sample the initial probability vector and the generating matrix α^i and S^i, respectively, following Algorithm 1; additional details about this algorithm are provided later in this section. Once α^i and S^i are sampled, we assign their values according to their designated locations in α and S. We note that for S the assignments are straightforward, while in α the values are multiplied by $P[i]$. This is due to the fact that when sampling class i we get α^i, which is summed up to 1. However, the total probability of going into class i should be $P[i]$. We verify that the expected value of the service time under α and S is finite.

In what follows we provide details of how to sample class i with Algorithm 1. We first randomly classify each state to to be either fully-absorbing, partially-absorbing or non-absorbing and assign them to $FullA_i$, $PartA_i$ and $NonA_i$, respectively, while asserting at least state being in $FullA_i$ or $PartA_i$. We next sample the $(Size_i, Size_i)$ transition matrix $Trans$ and adjust its values to reflect state classification.

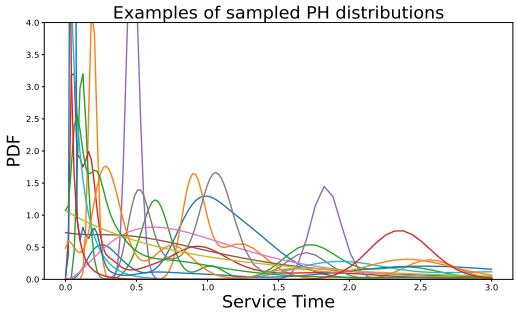

Figure 2: An example of 15 service time distributions sampled using our sampling method.

Fully absorbing states immediately transition to the absorbing state, thus the only non-zero value is on the diagonal. For normalizing the rates of partially-absorbing and non-absorbing states, $S^i[j,k]$ is being multiplied by a constant $-S^i[j,j]/\sum_{j=1,j\neq k}^{Size_i} S^i[j,k]$, which ensures that the sum of all non-diagonal values will sum to the diagonal value. For partially-absorbing states we also multiply by $1-p_j$, whereas, p_j is the probability that state j will reach the absorbing state upon departure. Once S^i is determined we sample α^i according to a Dirichlet distribution that allows for both unbalanced and balanced realizations.

3.1.4 Empirical Illustration of the Sampling Technique

In this section we empirically demonstrate that our approach indeed produces a set of phase-type distributions that provides "good coverage" of the space of all positive-valued distributions. For the sake of illustration, and without loss of generality, we scale all service time distributions to the mean value of 1. We generate a data set containing $60,000$ samples, which will later be used as our test set.

What should a sample providing "good coverage" of the set of all positive-valued distributions look like? We are unaware of any theoretical measures for such coverage. However, since positive-valued distributions with mean 1 will include cases with multiple modes, large mass concentrations at different locations, etc., we expect that for any given positive value, some distributions will concentrate most of the weight below this value while others above it. This implies that for any percentile, the value corresponding to this percentile will be very close to 0 for some distributions and very high for others. Thus, a large difference between minimum and maximum values across all percentiles can be taken as an (informal) indicator that the sample indeed provides a sufficient coverage for the set of all positive-valued distributions.

Table 1 presents the range of the service times for 6 different percentiles ranging from 25% to 99.9%. For each percentile level we provide the minimum and maximum service times in our data. For example the "Lower" value of $8.511e^{-6}$ and the "Upper" value of 1.0715 in the $25-$th percentile column shows that we have both, distributions that concentrate 25% of the mass very close to 0, and distributions with 75% of the mass above the value of 1 (recall that the mean is 1, so the latter indicates very high concentration above the mean). A similarly wide range is observed for other percentiles. Figure 2 shows a similar picture. Specifically, the figure presents 15 randomly selected PDFs from our sample. We observe a wide diversity of shapes, multi-modality, etc. Thus, we conclude that our sampling procedure indeed generates a diverse set of distributions. For comparison we also sampled $60,000$ PH distributions with a mean value of 1 using the procedure presented in (Horváth and Telek 2019). Their algorithm resulted in a much narrower range at every percentile. For example the 25^{th} and 99.9^{th} percentiles ranged from 0.28 to 0.3 and from 6.8 to 7.1, respectively. Thus, all distributions sampled with this method are very similar, which is insufficient for our needs.

Table 1: Range of percentiles for sampled service time PH distributions.

Percentile	25%	50%	75%	90%	99%	99.9 %
Lower	$8.5111e^{-6}$	0.0003	0.0027	0.0308	0.8391	1.1184
Upper	1.0715	1.5812	3.0376	7.7300	29.9446	29.9871

3.2 Generating Output

To generate the output, we use the arrival rate and PH representation of service time distribution (per input) to compute the first l components of the steady-state probability vector using the QBD method (Neuts (1981)), which is faster than other analytical methods for the case of PH-distributed service times. Since we feed the computed distribution to the neural network as a fixed vector, we truncate our computation at l such that the total probability of having more than l customers is smaller then ε (both l and ε are hyper-parameters). In our empirical evaluation we used $\varepsilon = 10^{-7}$ for which we achieved the desired performance with $l = 500$ in all generated samples. This corresponds to covering the probability of between 0 and 499 customers in the system; the last value of the output vector is set to the probability of having 499 or *more* customers. In terms of runtimes, sampling a single service time distribution and computing the output (which takes roughly 95% of the time) takes on average 0.56 seconds using Intel Xeon Gold 5115 Tray Processor with 128GB RAM.

4 THE NEURAL MODEL

In this section, we provide details into our neural model including input pre-processing, our choice of network architecture, and the loss function.

4.1 Input pre-processing

Once the sampling procedure is completed we receive pairs of arrival rate and service time distributions in the form of (α, S). Next, we compute the first n moments of the service time distribution. While these moments, together with the arrival rate could be used as an input to the neural network, we go through another pre-processing step as follows. It is a common practice in neural network models to normalize the inputs so that all input values will be within the same range. In our case, this means enforcing range similarity both between samples and within a sample (i.e., arrival rate and service moments). While scaling all service time distributions to the expected value is 1 (w.l.o.g), achieves between-sample similarity, as we increase the moments order they increase exponentially, thus violating within-sample similarity. Thus we apply a *log* transform to all service time moments, resolving this issue. Since the first moment in all cases is 1, we omit it from the input.

4.2 Network Architecture

We deploy a Fully-Connected (FC) feed-forward neural network. While other architecture types, such as Convolution Neural Nets (CNN) and Recurrent Neural Nets (RNN) were considered, we concluded that neither one was appropriate for this case: CNN is not well suited to our needs as the input size is only a small vector, while RNN is more suited for time-dependant, rather than stationary, systems. Thus, our **neural architecture** comprised a fully connected neural network with 6 hidden layers and a total of 274,269 parameters. We used the Rectified Linear Unit (ReLU) as the activation function for all hidden layers. Furthermore, we applied the Softmax function to guarantee that the output-layer weights sum up to 1. We note that in terms of computational complexity, the network we used is considered to be small (a large network typically contains several millions parameters). This leaves room for adding network complexity as our machine learning methodology is extended to more complex queueing systems. Finally, in the training phase, we use different learning rate functions, batch sizes and optimisers into to fine-tune the model.

Table 2: Neural network model performance for different number of service time moments.

# Moments	2	3	4	5	6	7	8	9	10	15	20
SAE	0.0165	0.005	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001

4.3 Loss function

For the training batch size B (a hyper-parameter), let Y and $\hat{Y}$ represent a matrix of size (B,l) of the true values of the stationary queue length distribution and the network prediction, respectively. Our loss function $Loss(\cdot)$ is given by

$$Loss(Y,\hat{Y}) = \frac{1}{B}\sum_{i=1}^{B}\sum_{j=0}^{l-1}|Y_{i,j}-\hat{Y}_{i,j}| + \frac{1}{B}\sum_{i=1}^{B} max_j(|Y_{i,j}-\hat{Y}_{i,j}|), \tag{1}$$

where the values $Y_{i,j}$ and $\hat{Y}_{i,j}$ refers to the i^{th} sample in the batch and the probability of having j customers in the system for the respective distributions. Note that the loss function refers to a single batch B.

From Equation (1) we observe that the loss function comprises two terms. The first term aims to minimize the overall absolute distances between the true and predicted distributions, while the second term seeks to minimize the maximum distance. This functional form is motivated by typical queueing applications where few smaller errors are preferred over a single large one (particularly since large errors tend to occur in the tail, which may be particularly important for providing service level guarantees). While there are many other ways to measure the distance between two distributions, we deemed them to be less suitable for our purposes. For instance, the popular KL-divergence is less suitable for distributions having numerous zero values, which is quite common in stationary queue-length distributions.

5 MODEL TUNING AND EXPERIMENTAL EVALUATION

5.1 Experimental Setting

In this section we discuss the different data-sets used to train and validate our model, the training procedure, the metrics used to evaluate our model, and the architecture and values of hyperparameters for our final model.

Data generation was performed via the sampling procedure described in Section 3.1. We produced three different data sets: the training, validation, and test data sets with sizes $1,200,000$, $60,000$ and $60,000$, respectively. Training the neural network (namely, its weights) was performed on the training set, while the validation set was used during training for the purpose of tuning hyper-parameters of the network. Lastly, model performance was assessed based on the test data set.

Model training and tuning was performed using the following sequence of steps. Once all three data-sets were sampled and pre-processed, including the computation of the output, we trained our model for every value of the number of service distribution moments $n = 2,..,20$. For each n, we fine-tuned our model, i.e., chose the best hyper-parameters (e.g., learning-rate schedule, optimizer), based on our validation set. Different hyper-parameters settings were compared via a simple metric, *SAE* (i.e., Sum Absolute Error), given by the following equation:

$$SAE(Y_i,\hat{Y}_i) = \sum_{j=0}^{l-1}|Y_{i,j}-\hat{Y}_{i,j}|,$$

where Y_i and $\hat{Y}_i$ correspond to two matrices of vectors l of the true values of the i^{th} sample in the test set stationary queue-length distribution and of the predicted values, respectively. Note that *SAE* is the absolute sum of errors (i.e., absolute distances between the true and the estimated distributions) computed for each sample and then averaged over the entire validation data set. The main advantages of the metric is that it has a straightforward probabilistic meaning and it covers the entire PDF of the stationary queue length.

Table 3: Neural network model performance on Test Data.

Percentile	25%	50%	75%	90%	99%	99.9 %
Metric 1	0.000220	0.0004272	0.00087	0.002001	0.008757	0.02001
Metric 2	0.000342	0.000718	0.00094	0.000897	0.00148	0.00412

Also, for each sample it provides an upper bound on the maximum difference between the two cumulative distribution functions (CDFs), which has statistical justification when examining distances between two distributions, e.g., in the Kolmogorov–Smirnov test. For each value of n we chose settings that minimizes the value of SAE. Finally, we searched for the 'best' value of n, i.e., the value that achieves the minimum of SAE over the validation set. **The number of moments** was selected by computing the average value (over the test set) of SAE as a function of the number of moments.

Table 2 presents the results of the service moment analysis. Specifically, convergence is achieved once the number of moments reaches $n = 5$. This implies that only the *first five* service time moments appear to determine the dynamics of the $M/G/1$ queue. This result is somewhat surprising as two distributions sharing the same first five moments may differ significantly, yet will, apparently, result in the same stationary queue-length distribution. On the other hand, as discussed earlier, the average queue length is known to be determined by only the first two service time moments, which makes this result more sensible.

Once the value of n has been determined to be 5, we fixed the values of all other hyper-parameters. We used the Adam optimization algorithm for an iterative update of the network weights based on the training data. The corresponding weight decay value for avoiding over-fitting was 10^{-5}, while the selected training batch size was 128. The learning rate decayed exponentially, starting from 0.01. The number of trained epochs was 300. The model was trained on an NVidia A100 GPU. For the sizes of training, validation, and test data sets provided above, sampling the entire data took less than 7 hours.

5.2 Results

Throughout this section we show the results of the "best" network with $n = 5$, tuned as described above. For each sample in the test set we compare the true stationary queue length distribution to its counterpart predicted by our model. In addition to SAE, we also define the following additional metric which focuses on the inverse of the CDF of the number of customers in the system. Specifically, we compute the average relative error for each of the 6 different percentile values (which we refer as EP - Error Percentile), namely, $25\%, 50\%, 75\%, 90\%, 99\%, 99.9\%$, as follows:

$$EP(Y, \hat{Y}, percentile) = \frac{1}{N} \sum_{i=1}^{N} \frac{F_{Y_i}^{-1}(percentile) - F_{\hat{Y}_i}^{-1}(percentile)}{F_{Y_i}^{-1}(percentile)},$$

where F in the stationary distribution of the number of customers in the system and N is the size of the test set. This metric is particularly effective in detecting differences in the tails of the two distribution functions.

Table 3 presents the performance of our model under both metrics. Note that the interpretation of values for SAE and EP are quite different. For SAE (the second row of the table) the values represent the exact percentiles of the absolute difference between true and predicted distributions over the test data set. For example, the 99.9-th percentile was only .02, indicating a high correspondence between the two distributions. The overall value of SAE is 0.001, indicating an excellent fit.

For EP (last row of the table) the values represent the average relative absolute differences between the true and the predicted distribution at each percentile level over the entire test set. For example, the largest relative difference was only 0.412% achieved at the 99.9-th percentile. As demonstrated in Table 3 the errors are very small, where in 99% of the cases the error is smaller than 0.01 implying that our predictions are very reliable. Furthermore, as the results for EP suggest, the predicted distribution produced by our model remains accurate at the tail as well, since the average error for all percentiles is low.

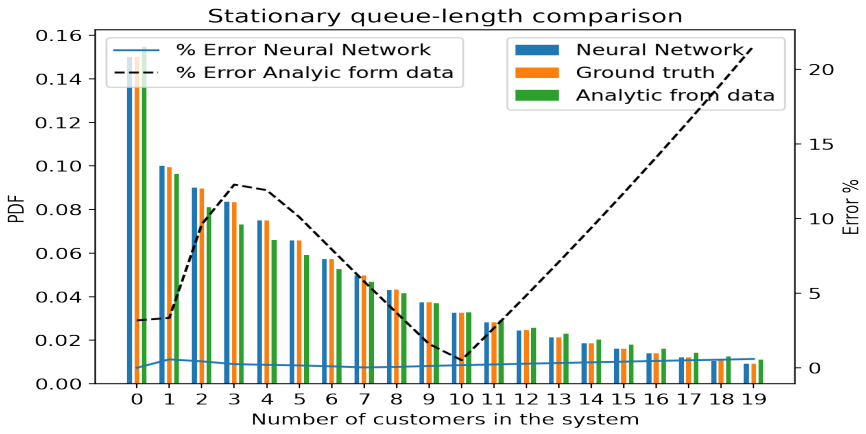

Figure 3: Comparing our ML approach and to the analytic method.

Overall our results indicate that neural predictions of stationary queue-length distribution are very accurate. However, it should be pointed out that this evaluation was performed on our test data where the required inputs, i.e., the first five moments of the service distribution, were known with certainty. In the following section we test out model under more realistic conditions, based on empirical data set.

6 MODEL EVALUATION ON EMPIRICAL DATA

In most realistic settings, only an empirical set of data observations for service times and arrivals is likely to be available. To test our approach in such settings, instead of having a closed-form expressions for the inter-arrival and service time distributions, we assume that we only have access to the timestamps of arrivals, service commencements, and service completions. Given such data, and assuming the system is known to operate as an $M/G/1$ queue, there are two approaches for obtaining the stationary distribution, namely the traditional "analytical approach" and the "neural net" approach of the current paper.

Under the analytical approach, we first use the data to fit some parametric distribution to service times (we shall use PH), and then apply one of the available closed-form expressions to obtain the stationary distributions (here, we shall use the QBD method). The neural net approach (assuming a pre-trained network is available) is simpler: we only need to compute the first five service time moments from the data, and then we apply the neural network to predict stationary distribution.

Both methods will incur some errors. In the analytic approach the main accuracy loss can be expected when fitting the service time distribution, while numerical errors of QBD calculation are controllable and typically extremely low. In the neural net method there are two potential sources of error: estimation of the service time moments from the data, and prediction error of the neural network model.

Keeping this in mind, the case-study was executed as follows: (1) We sampled a service distribution using our sampling method described above (Section 3.1), set the service rate to 0.85, and compute the corresponding stationary queue length distribution. This provides us with the "Ground truth". (2) Next, we sampled $50,000$ data points from the service time distribution producing an "observed" sample. (3) We applied the analytic approach to the data by employing a standard EM method to fit the distribution, followed by the QBD method. This gave us the "Analytic from data" estimate. The best practice is to use the EM algorithm to fit the closest PH distribution, see (Fackrell 2009). (4) We applied our approach producing the "Neural Network" estimate. For the neural approach the first five moments were fitted by employing an unbiased estimator directly from the data. That is, $\bar{X^i} = \sum_{i=1}^{N} \frac{X^i}{N}, i = 1,\ldots,5$, where X is the service time, and N is the sample size. Note that the runtime the EM algorithm of the "Analytic from data" solution depends on the maximum size of the PH distribution fitted to data. To keep runtimes reasonable, PH sizes are limited 20.

Table 4: Service moments and runtimes under the CPU.

	1	2	3	4	5	pre-processing (sec)	Running time
Analytic (EM)	0.994	2.705	10.093	44.131	217.936	4320	0.16
Neural Netowrk	0.994	2.398	7.037	21.961	71.227	0.3	0.12
Ground Truth	1	2.418	7.106	22.19	71.97	0	0.44

Table 4 presents the first five moments of service times estimated by each method (for neural nets these were computed directly from the empirical data, for the analytic approach they were computed from the PH distribution obtained by the EM method). We see that, unsurprisingly, the direct estimates used by the neural net are far more accurate, particularly with respect to higher-order moments. This points to substantial differences between the service time distribution estimated by the EM method and the ground truth distribution from which the data was generated. Table 4 also shows both the pre-processing runtimes (for the analytic approach this is the running time of the EM method, for the neural network this is the time to compute the first five moments) and the algorithm runtimes (QBD computation times for the analytic and ground truth approaches; network computation time for the neural network approach). The largest difference is in the extremely long pre-processing time for the analytic approach compared to near-instantaneous time it takes to pre-process for the neural net.

Next, we compare the accuracy levels of the two approaches. First we note that the value of *SAE* for the analytic method is 0.094, while it is 0.002 for the neural net, indicating a significantly higher average accuracy for the latter. For a more detailed comparison we plot the steady-state probabilities of i number of customers in the system for $i = 0, \ldots, 19$ (see Figure 3). For each i we plot the true value, as well as the estimates obtained by the two methods. On the second y-axis we plot the relative errors for each i for the two methods. We observe that while the estimates obtained by the neural net are very accurate, with relative error never exceeding 0.6% (the average error is 0.3%), the errors for the analytic approach reach as high as 12% even for low values of i where most probability mass is concentrated. For $i \geq 10$ the relative errors increase at a linear rate, reaching over 55% for $i = 19$ (this increase continues at the same rate for higher values of i not shown on the plot). While the relative errors for the neural network also increase with i, the increase is much more gradual, reaching only 0.6% at $i = 19$. Overall, we conclude that the neural network produces far better estimates of the steady state distribution than the analytic technique.

7 CONCLUSION & FUTURE WORK

In this paper, we applied neural nets to devise a new data-driven solution to a fundamental problem in queueing theory. Our results show that machines can accurately learn these distributions using only a relatively compact neural network architecture. Surprisingly, the accuracy does not significantly improve when we go beyond the 5^{th} moment of general service time distributions. We also provided an evaluation using empirical data, thus demonstrating the superiority of our approach to analytical methods both in terms of accuracy and runtime.

Our results motivate future studies in which more complex queueing models will be considered. However, this leads to two challenges: (1) ground truth for more complex models is not available, and (2) it is unclear how to obtain large and reliable data sets on which the neural net can be trained. Below, we briefly discuss potential solutions, while distinguishing between two data types, namely synthetic and real-world datasets.

Synthetic data is sampled from a queueing system with a known structure, as we have done in this paper. However, since we are typically interested in applying neural models to systems for which an analytical solution does not exist, producing the target on which the model can be trained is not straightforward. One workaround is to train the network on special cases where the solution is within reach. For example, an immediate extension would be to use our PH sampling algorithm to produce synthetic data for systems where arrival and/or service processes are of PH form (e.g., $PH/PH/1$) - such systems are analytically

tractable and an extension to $G/G/1$ type queues should be feasible. Alternatively, one could employ simulation models to generate training samples.

Real data describes the queueing dynamics of the system can be extracted from databases of many organizations. The advantage of the real data is that "ground truth" of actual system performance is directly available. The main problem is that one cannot typically generalize from one system to the other. Moreover, different time periods for the same system may not share the same distributions of service times or inter-arrival times, as the data essentially captures a single realization for a single queuing system. This issue can potentially be addressed by combining various data sets (or data slices) together when fitting the distributions and learning the neural model.

REFERENCES

Asmussen, S. 2003a. *Applied Probability and Queues*. New York, NY: Springer New York.

Asmussen, S. 2003b. *Random Walks*. New York, NY: Springer New York.

Fackrell, M. 2009. "Modelling healthcare systems with phase-type distributions". *Health Care Management Science* (12):11–26.

Horváth, G., and M. Telek. 2019. *Markovian Performence Evaluation with BuTools*, 253–268. New York, NY: Springer.

Kyritsis, A. I., and M. Deriaz. 2019. "A Machine Learning Approach to Waiting Time Prediction in Queueing Scenarios". In *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*. July 4th-7th, California, USA, 17-21.

LeCun, Y., Y. Bengio, and G. Hinton. 2015, May. "Deep learning". *Nature* 521(7553):436–444.

Neuts 1981. *Matrix-Geometric Solutions in Stochastic Models.* Berlin, Germany: Springer Berlin.

Nii, S., T. Okudal, and T. Wakita. 2020. "A Performance Evaluation of Queueing Systems by Machine Learning". In *2020 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan), Taoyuan, Taiwan*, 1–2.

Okamura, H., and T. Dohi. 2015. "mapfit: An R-Based Tool for PH/MAP Parameter Estimation". In *Quantitative Evaluation of Systems*, edited by J. Campos and B. R. Haverkort, 105–112. Cham: Springer International Publishing.

Palomo, S., and J. Pender. 2020. "Learning Lindley's Recursion". In *Proceedings of the Winter Simulation Conference*, edited by G. Bae, S. Lazarova-Molnar, Z. Zheng, B. Feng, and S. Kim, WSC '20, 644–655: Institute of Electrical and Electronics Engineers Press.

Pender, J., and E. Zhang. 2021. "Uniting Simulation and Machine Learning For Response Time Prediction In Processor Sharing Queues". In *2021 Winter Simulation Conference (WSC)*, edited by S. Kim, B. Feng, K. Smith, and Z. Z. S. Masoud. Institute of Electrical and Electronics Engineers.

Sherzer, E., A. Senderovich, O. Baron, and D. Krass. 2022. "Can machines solve general queueing systems?". https://arxiv.org/abs/2202.01729.

Telek, M., and G. Horváth. 2007. "A Minimal Representation of Markov Arrival Processes and a Moments Matching Method". *Performance Evaluation* 64(9):1153–1168.

Vazquez-Avila, J. L., and R. Sandoval-Arechiga. 2019, Jul. "A Fast Simulation Model Based on Lindley's Recursion for the G/G/1/K Queue". *Mathematical Problems in Engineering* 2019:346–389.

Xiao, W., and R. Bhardwaj. 2018. "Gandiva: Introspective Cluster Scheduling for Deep Learning". In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 595–610.

AUTHOR BIOGRAPHIES

OPHER BARON is a Distinguished Professor of Operations Management and the Academic Director, MMA Program at the Rotman School of Management, University of Toronto. His email address is opher.Baron@rotman.utoronto.ca.

DMITRY KRASS is a Professor of Operations Management and Statistics and the Sydney C. Cooper Chair in Business and Technology at the Rotman School of Management, University of Toronto. His email address is krass@rotman.utoronto.ca.

ARIK SENDEROVICH is an Assistant Professor at the School of Information Technologies at York University. His email address is sariks@yorku.ca.

ELIRAN SHERZER is a postdoc at the Rotman School of Management, University of Toronto. His email address is eliran.sherzer@utoronto.ca.