NIM: GENERATIVE NEURAL NETWORKS FOR MODELING AND GENERATION OF SIMULATION INPUTS

Emily A. Herbert
Wang Cen
Peter J. Haas

College of Information and Computer Sciences
University of Massachusetts Amherst
140 Governors Drive
Amherst, MA USA
{emilyherbert, cenwang, phaas}@cs.umass.edu

ABSTRACT

We introduce Neural Input Modeling (NIM), a generative-neural-network framework that exploits modern data-rich environments to automatically capture complex simulation input distributions and then generate samples from them. Experiments show that our prototype architecture NIM-VL, which uses a variational autoencoder with LSTM components, can accurately, and with no prior knowledge, automatically capture a range of stochastic processes, including mixed-ARMA and nonhomogeneous Poisson processes, and can efficiently generate sample paths. Moreover, we show that the outputs from a queueing model with (known) complex inputs are statistically close to outputs from the same queueing model but with the inputs learned via NIM. Known distributional properties such as i.i.d. structure and nonnegativity can be exploited to increase accuracy and speed. NIM has the potential to help overcome one of the key barriers to simulation for non-experts.

Keywords: generative neural network, input modeling, stochastic process generation, LSTM, VAE

1 INTRODUCTION

Stochastic discrete-event simulation is a time-honored technology for improving the design and operation of complex engineered systems under uncertainty, but the barriers to entry are high. Although modern simulation software tools provide graphical interfaces that can greatly ease the task of specifying the simulation model structure, modeling the simulation inputs remains one of the most challenging tasks for a non-expert. Our goal is to facilitate this process via automation.

Traditionally, data for fitting input distributions has been expensive and painful to collect—e.g., a human would have to stand on a factory floor, stopwatch in hand—and hence has been in short supply. With little data available, a modeler typically imposes strong simplifying assumptions, for example, by assuming that the interarrival times to a system are independent and identically distributed (i.i.d.) according to one of a set of supported distribution functions from which the system can efficiently generate samples. I.i.d. distributions with complex features such as multimodality or phase-type structure (Neuts 1981) are typically hard to capture, however, and fidelity is sacrificed. The situation becomes even more challenging when an interarrival sequence is not well modeled as a sequence of i.i.d. random variables. In this case, with little guidance or software support, the user faces a bewildering array of possible models for autocorrelated, possibly non-

SummerSim-WIP, 2019 July 22-24, Berlin, Germany;

stationary interarrival sequences, including time series models such as ARIMA, GARCH, SETAR, and so on, with various choices for the innovations distribution (Box, Jenkins, Reinsel, and Ljung 2016), or direct point process models of arrival times such as nonhomogeneous, compound, clustered, or doubly-stochastic Poisson processes (Cox and Isham 1980). Even after settling on a stochastic-process model, efficiently generating sample paths can be decidedly nontrivial.

The other option for non-expert input modeling is to fit an empirical distribution (ED) for i.i.d. data and use input traces (IT) for more general stochastic processes. Both of these approaches are problematic for a number of reasons. Both ED and IT approaches suffer from the fact that the data values that can be produced during a simulation run are strictly limited to those in the available data. This issue is one aspect of a general overfitting problem in which the simulation model captures the training data precisely but does not generalize well beyond this data. The IT approach has the additional drawback that, if the simulation model is to be deployed widely, then the need to move potentially large amounts of data around is cumbersome and raises potential privacy issues.

We aim to exploit the fact that data is becoming ubiquitous due to the increasing use of sensors, the emergence of the Internet of things (IoT), and the retention of log data in formally defined process management systems (van der Aalst 2018). Other potential sources of structured log data include information extraction from text (Niklaus, Cetto, Freitas, and Handschuh 2018), as well as from images and video (Zhou, Xu, and Corso 2018). Our key observation is that, in data-rich environments, neural networks are a powerful and flexible tool for learning complex and subtle patterns from data, and therefore a promising means for automating the tasks of learning simulation input distributions and of generating samples from these distributions during simulation runs.

We introduce NIM (Neural Input Modeling), a framework for automated modeling and generation of simulation input distributions. NIM uses *generative neural networks* (GNNs) which not only learn a complex statistical distribution while avoiding overfitting, but provide a means of sampling from the distribution as well. We first give a brief overview of our NIM prototype. We then show empirically how NIM can accurately and automatically capture complex stochastic input process and then simulate them. We then show the effectiveness of NIM in the context of a queueing simulation model with complex inputs. To our knowledge, this is the first attempt to use GNNs in order to automate input modeling, thereby helping to democratize the use of stochastic simulation methodology.

2 NIM OVERVIEW

Our initial NIM prototype uses a particular form of GNN called a *variational autoencoder* (VAE); see (Doersch 2016, Kingma and Welling 2013) for derivations and details. A VAE uses a pair of neural networks to learn an internal representation of a stochastic process from data (the encoder) and then transform a sequence of i.i.d. Gaussian input variables into a realization of the modeled process (the decoder). A VAE does not need to make any prior assumptions about the features of the training data. In our prototype, we designed both the encoder and the decoder to contain a Long Short-Term Memory (LSTM) layer (Hochreiter and Schmidhuber 1997). Use of LSTM layers allows for concise capture of time-dependent features when encoding the training data. We refer to the resulting neural architecture as NIM-VL.

Training Procedure: The NIM-VL architecture varies slightly depending on whether the model is being trained or is generating data. Figure 1 shows the architecture used for training. Each observed sample path $\mathbf{x} = (x_1, \ldots, x_t)$ in the training data is passed through the encoder E to produce $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}$. Next, we set

$$z_i = \tilde{\sigma}_i \xi_i + \tilde{\mu}_i \tag{1}$$

for $i \in [1..t]$—where $\xi_i, \ldots, \xi_t$ are i.i.d. $N(0,1)$ random variables—to produce the internal representation $\mathbf{z}$. Next, $\mathbf{z}$ is concatenated with a shifted $\mathbf{x}$ to produce Z, which is then passed through the decoder D to produce

$\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$. During generation, these latter two vectors will be used to generate independent normal samples using a transformation almost identical to (1). The key idea is to (i) design D so that, given i.i.d. $N(0,1)$ random variables $z_1,\ldots,z_t$ and data $x_1,\ldots,x_{t-1}$ from the target distribution, arranged as in Z, the decoder will produce $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$ such that the resulting normal samples will jointly be distributed as a sample of the target stochastic process, and (ii) design E so that $z_1,\ldots,z_t$, taken together, look like i.i.d. samples from a standard normal distribution $N(0,1)$, since this is what is needed during generation.

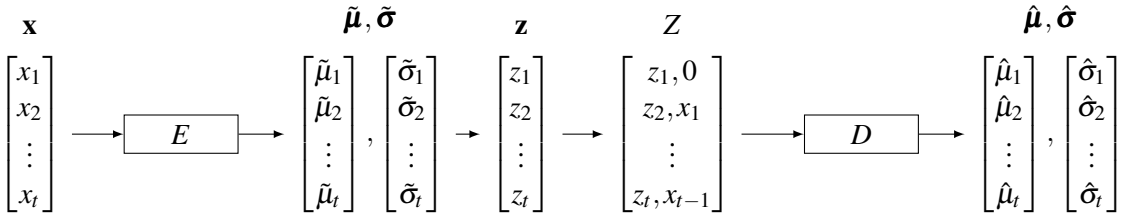


Figure 1: NIM-VL training architecture.

As the observed sample paths in the training data are fed into the training network, the weights in the two LSTMs are simultaneously trained via backpropagation (basically gradient descent) to minimize the loss function

$$L(\mathbf{x},\tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}},\hat{\boldsymbol{\mu}},\hat{\boldsymbol{\sigma}}) = -\sum_{i=1}^{t}\big(\log\tilde{\sigma}_i^2-\tilde{\mu}_i^2-\tilde{\sigma}_i^2+1\big)+\sum_{i=1}^{t}\big(\log 2\pi+\log\hat{\sigma}_i^2+\frac{(x_i-\hat{\mu}_i)^2}{\hat{\sigma}_i^2}\big).$$

The first term represents the KL-divergence between $N\big(\tilde{\boldsymbol{\mu}},\mathrm{diag}(\tilde{\boldsymbol{\sigma}})\big)$ and $N(\mathbf{0},\mathbf{I})$; minimizing this term helps achieve goal (ii) above. The second term is the negative log-likelihood of $\mathbf{x}$ under the $N\big(\hat{\boldsymbol{\mu}},\mathrm{diag}(\hat{\boldsymbol{\sigma}})\big)$ distribution; minimizing this term (i.e., maximizing the log-likelihood), helps achieve goal (i), which is to make the synthetic data look like the training data. Note that the KL-divergence term acts as a regularizer and helps prevent overfitting to the training data.

Generation procedure: After the VAE network is trained, the synthetic-data generator essentially runs the decoder as in Figure 1 starting with Z and then generating synthetic data as normal samples with parameters $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$. In Z, the $\mathbf{z}$ variables now comprise actual i.i.d. $N(0,1)$ samples, and the role of $\mathbf{x}$ is now played by the synthetically generated data $\mathbf{y}=(y_1,\ldots,y_t)$, which must be generated via t iterative steps.

In more detail, Figure 2 shows the architecture for generating synthetic data.

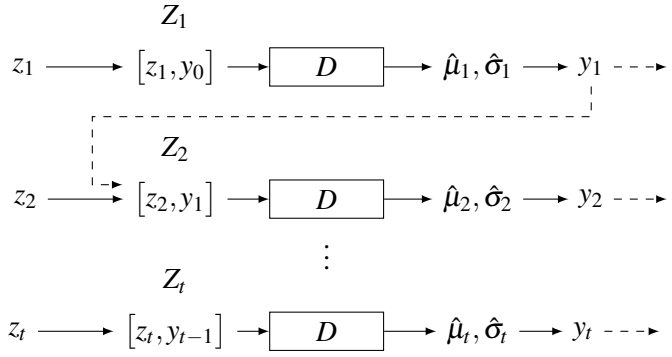


Figure 2: NIM generation architecture.

At iteration step i, the generator draws variate $z_i\sim N(0,1)$ and combines it with y_{i-1} to create Z_i, where $y_0=0$. Then Z_i is passed through D, producing $\hat{\mu}_i$ and $\hat{\sigma}_i$. Finally, the generator computes $y_i=\hat{\sigma}_i w_i+\hat{\mu}_i$, where $w_i\sim N(0,1)$. This procedure is repeated until y_t is found and $\mathbf{y}=(y_1,y_2,\ldots,y_t)$ is collected.

If it is known a priori that the target random variables are positive, then we can improve accuracy by transforming the training data $\mathbf{x}$ via $x_i' = \log x_i$ and then transforming the resulting synthetic data $\mathbf{y}'$ via $y_i = e^{y_i'}$. Similar techniques can handle upper bounds and paired lower/upper bounds. If the target variables are known to be i.i.d., then each of the two LSTMs can be replaced by a single fully-connected layer, yielding a simple multilayer perceptron (MLP) with an input, hidden, and output layer. This simpler architecture, called NIM-VM, increases both accuracy and generation speed.

3 INITIAL EXPERIMENTS

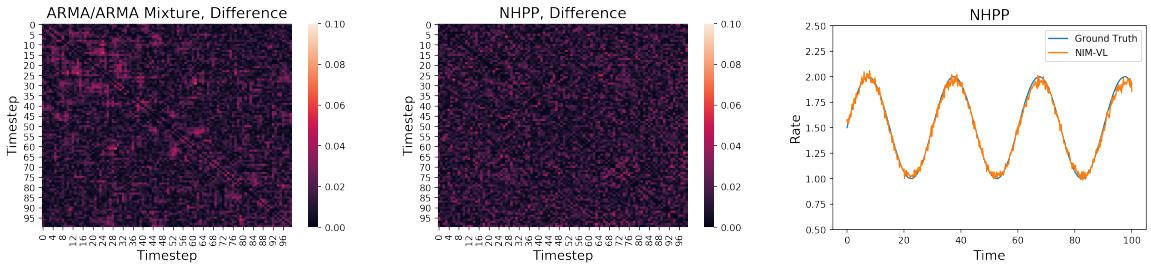
To indicate the potential of our NIM technology, we report results from several initial experiments.

3.1 Complex Input Processes

We tested NIM-VL on two complex, nonstationary stochastic processes: a nonstationary ARMA/ARMA mixture process and the interarrival time sequence for a nonhomogenous Poisson process. In each case, The NIM-VL model was trained with 10,000 ground truth sample paths, each of length 100. (We refer to sample paths that are generated traditionally as "ground truth" and sample paths generated by our VAE as "NIM-VL".) We used 64 LSTM units and 128 hidden nodes in each of the encoder and decoder.

The trained model is then used to generate $10,000$ NIM-VL sample paths of length 100, and these NIM-VL sample paths are compared against $10,000$ validation ground truth sample paths (which are distinct from the ground truth sample paths used for training). As a simple way to compare these complex, nonstationary stochastic processes, we take the validation ground truth sample paths $\mathbf{x}_n = (x_{n,1}, \ldots, x_{n,100})$ for $n = 1, \ldots, 10{,}000$ and compute empirical correlation coefficients $\hat{\rho}_{ij}^{\text{GT}} = \widehat{\text{Corr}}[X_i, X_j]$ for $1 \le i, j \le 100$. We similarly compute $\hat{\rho}_{ij}^{\text{NIM}}$ for the NIM-VL sample paths, and plot the differences $d_{ij} = |\hat{\rho}_{ij}^{\text{NIM}} - \hat{\rho}_{ij}^{\text{GT}}|$. The detailed results are given below.

ARMA/ARMA mixture: We first consider a mixture $\{X_i\}_{i\ge1}$ of two nonstationary ARMA(2,2) processes $\{A_i\}_{i\ge1}$ and $\{B_i\}_{i\ge1}$ (with standard Gaussian innovations). We run both processes in parallel; at time i, we set $X_i = A_i$ with probability 0.5 and otherwise set $X_i = B_i$. The parameters of the two processes are $(0.95, -0.1; 0.2, 0.95)$ and $(0.8, -0.3; 0.3, 0.7)$. Figure 3a gives the correlation difference plot; the largest absolute correlation difference value is 0.0609, indicating good agreement.



(a) (b) (c)

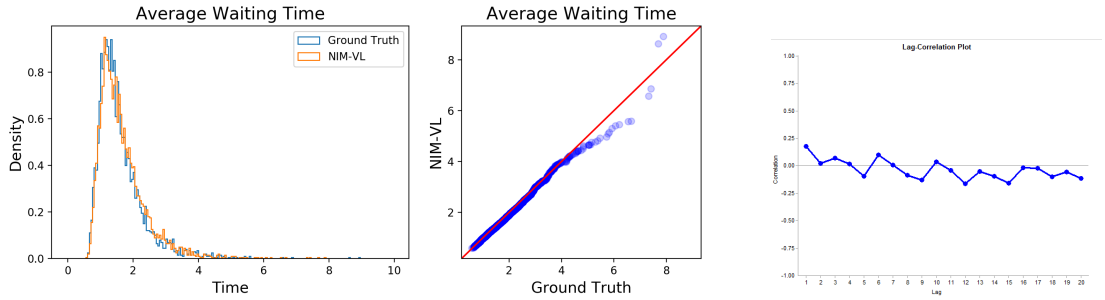
Figure 3: Ground truth vs NIM-VL for ARMA/ARMA mixture and nonhomogenous Poisson processes.

Nonhomogeneous Poisson process: We next test our approach on a nonhomogenous Poisson process (NHPP) interarrival time process with a rate function $\lambda(t) = \frac{1}{2}\sin(\frac{\pi}{8}t) + \frac{3}{2}$. Because we know a priori that interarrival times for a NHPP are positive with probability 1, we apply the transformation technique described in Section 2 when using NIM-VL. Ground truth data is generated via thinning (Lewis and Shedler

1979). Figure 3b gives the correlation difference plot; the largest absolute correlation difference is 0.0550. To further compare ground truth and VAE sample paths, we compared the empirical arrival rate function to the ground truth function $\lambda(\cdot)$ given above. Specifically, for each NIM-VL sample path, we computed the sequence of arrival times by taking partial sums of interarrival times. We next divided the interval $[0,100]$ into subintervals of length 0.2, and computed the average number of arrivals in each subinterval, where the average was taken over the 10,000 sample paths. Figure 3c shows the resulting empirical arrival rate function plotted against $\lambda(\cdot)$. As can be seen, the agreement is quite good, further indicating the effectiveness of NIM at capturing the NHPP process (with no prior knowledge that it *is* an NHPP process).

3.2 A Queueing Simulation

Our final experiment examines the end-to-end effect of using inputs from NIM-VL to simulate the average waiting time $\overline{W}_{100}$ of the first 100 jobs in an NHPP/Gamma/1 FIFO queue; the interarrival time process is an NHPP as before and service times are i.i.d. Gamma$(1.2, 0.4)$. The training sets for interarrival times and for service times each comprise 1000 sample paths with 50 observations per path. We apply our log-transformation method. In Sections 3.2 and 3.3, we use 16 LSTM units and 32 hidden nodes for each of the encoder and decoder. Figure 4a shows the empirical density of $\overline{W}_{100}$ over 4,000 simulation replications. Figure 4b shows the Q-Q plot. Again, there is close agreement. Note that each sample path in the training data comprises only 50 customers, but we simulate 100 customers. This indicates that, if the system is not too nonstationary, we can extrapolate beyond our training set; a trace-driven simulation would not be applicable here. Note that if we simply feed, say, 100 NHPP interarrival times into distribution-fitting software (we used ExpertFit), and ignore the somewhat subtle departure from independence as shown, e.g., in a lag-correlation plot (Figure 4c), we would erroneously model interarrivals as i.i.d. Pearson Type VI random variables. If we noticed the lack of independence, then we would have no guidance on what to do next.



(a) Empirical density (b) Q-Q plot (c) NHPP lag correlation

Figure 4: Empirical distribution of $\overline{W}_{100}$ in an NHPP/Gamma/1 queue, plus NHPP lag-correlation plot.

3.3 Generation Speed

Because NIM is designed to handle generic input distributions, we would not expect it to outperform customized generation methods. It is reasonable to ask, however, whether NIM can generate realizations of stochastic processes rapidly enough to be of practical use. Our initial results are positive in this respect. For example, NIM-VL, as implemented in PyTorch on a commodity 2019 MacBook Pro, is able to generate 1,000 sequences of 1,000 learned NHPP interarrival times in roughly 0.85 seconds. For i.i.d. data, NIM-VM (with 32 hidden nodes and 0 LSTM units per encoder and decoder) is able to generate 10^6 i.i.d. learned exponential random variables in roughly 0.12 seconds. Note that that generation consists mainly of simple matrix multiplications, which can potentially be accelerated via specialized hardware such as GPUs.

4 CONCLUSION AND FUTURE WORK

Generative neural networks are promising tools for automating the modeling and generation of simulation input data. Our NIM prototype is able to automatically capture complex distributions and autocorrelation structure, thereby facilitating one of the hardest tasks in a simulation study. High-level properties that are known a priori can be exploited to improve speed and accuracy. NIM is not a silver bullet, however. Estimating the tail of a distribution from data, especially if the tail is heavy, is challenging for any input modeling scheme. Similarly, extrapolating to simulation lengths far beyond the training data can be problematic for highly nonstationary processes; sanity checking and validation are still needed.

In ongoing work, we are conducting an extensive experimental study and refining our accuracy metrics. We are also exploring techniques for handling discrete random inputs, marked point processes (Cox and Isham 1980), and—analogously to ARIMA models—input processes that are nonstationary but homogeneous in the sense of (Box, Jenkins, Reinsel, and Ljung 2016, p. 88). We aim to release NIM as an open source tool to facilitate the use of simulation in modern data-rich environments by both experts and non-experts.

REFERENCES

Box, G., G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. 2016. *Time Series Analysis: Forecasting and Control*. Wiley.

Cox, D. R., and V. Isham. 1980. *Point Process*. Chapman and Hall.

Doersch, C. 2016. "Tutorial on variational autoencoders". *arXiv preprint arXiv:1606.05908*.

Hochreiter, S., and J. Schmidhuber. 1997. "Long short-term memory". *Neural computation* vol. 9 (8), pp. 1735–1780.

Kingma, D. P., and M. Welling. 2013. "Auto-encoding variational Bayes". *arXiv preprint arXiv:1312.6114*.

Lewis, P. A. W., and G. S. Shedler. 1979. "Simulation of nonhomogeneous Poisson processes by thinning". *Nav. Res. Logist. Quart.* vol. 26, pp. 403–413.

Neuts, M. 1981. *Matrix-Geometric Solutions in Stochastic Models*. Dover.

Niklaus, C., M. Cetto, A. Freitas, and S. Handschuh. 2018. "A Survey on Open Information Extraction". In *COLING*, pp. 3866–3878.

van der Aalst, W. M. P. 2018. "Process mining and simulation: a match made in heaven!". In *SummerSim*, pp. 4:1–4:12.

Zhou, L., C. Xu, and J. J. Corso. 2018. "Towards Automatic Learning of Procedures From Web Instructional Videos". In *AAAI*, pp. 7590–7598.

AUTHOR BIOGRAPHIES

EMILY A. HERBERT is a Ph.D. student at the University of Massachusetts Amherst, College of Information and Computer Sciences. She completed her B.S. in Computer Science at Trinity University in 2018. Her email address is emilyherbert@cs.umass.edu and her web page is cs.umass.edu/~emilyherbert.

WANG CEN completed his M.S. in Computer Science at the University of Massachusetts Amherst in 2019 and his B.S.E. in Aeronautical and Astronautical Engineering from Shanghai Jiao Tong University in 2015. His email address is cenwang@umass.edu.

PETER J. HAAS is a Professor in the College of Information and Computer Sciences, and an Adjunct Professor in the Department of Industrial Engineering, both at the University of Massachusetts Amherst. His email address is phaas@cs.umass.edu and his web page is https://www.cics.umass.edu/people/haas-peter.