Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

NIM: GENERATIVE NEURAL NETWORKS FOR SIMULATION INPUT MODELING

Wang Cen
Emily A. Herbert
Peter J. Haas

College of Information and Computer Sciences
University of Massachusetts Amherst
140 Governors Drive
Amherst, MA 01003, USA

ABSTRACT

Neural Input Modeling (NIM) is a novel 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 automatically capture complex stochastic processes. Outputs from queuing simulations based on known and learned inputs, respectively, are also statistically close.

1 INTRODUCTION

Simulation input modeling remains one of the most challenging tasks for a non-expert. For example, when using observations of job arrival times to model the input process to a queue, simulationists often use distribution-fitting software to model interarrival times as 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. Distributions with complex features such as multimodality are typically hard to capture, however. The situation becomes even more challenging when an interarrival sequence is not well modeled as i.i.d. random variables. In this case, with little guidance or software support, the user faces a bewildering array of possible time series and point process models for autocorrelated, possibly nonstationary interarrival sequences. Even after settling on a stochastic-process model, efficiently generating sample paths can be decidedly nontrivial. Moreover, the use of empirical distributions for i.i.d. variates or input traces (IT) for more general stochastic processes is often problematic: they overfit to the training data and, for IT, model deployment can require moving potentially large amounts of data around, which is cumbersome and raises potential privacy issues.

We exploit the fact that data is becoming ubiquitous due to the increasing use of sensors, the emergence of the Internet of things (IoT), and annotation of text, audio, and video via machine learning. In such data-rich environments, neural networks (NNs) are a powerful and flexible tool for learning complex and subtle patterns from data. We propose the use of NNs for modeling and generation of simulation inputs.

2 NEURAL INPUT MODELING

NIM (Neural Input Modeling) is a novel framework for automated modeling and generation of simulation input distributions. Our initial NIM prototype uses a particular form of generative neural network called a *variational autoencoder* (VAE); see Doersch (2016) 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, and the training procedure guards against overfitting.

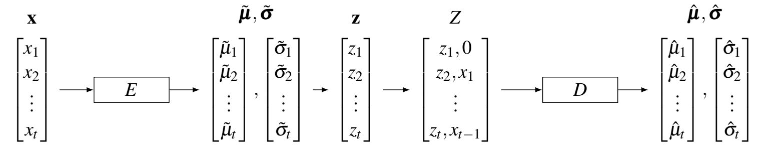
$\mathbf{x}$
$\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}$
$\mathbf{z}$
Z
$\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}$
E
D
$x_1, x_2, \vdots, x_t$
$\tilde{\mu}_1, \tilde{\mu}_2, \vdots, \tilde{\mu}_t$
$\tilde{\sigma}_1, \tilde{\sigma}_2, \vdots, \tilde{\sigma}_t$
$z_1, z_2, \vdots, z_t$
$z_1,0$; z_2,x_1; $\vdots$; z_t,x_{t-1}
$\hat{\mu}_1, \hat{\mu}_2, \vdots, \hat{\mu}_t$
$\hat{\sigma}_1, \hat{\sigma}_2, \vdots, \hat{\sigma}_t$

Figure 1: NIM-VL training architecture.

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 $z_i = \tilde{\mu}_i + \tilde{\sigma}_i \xi_i$ 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, we generate $\mathbf{z}, \boldsymbol{\nu} \sim N(\mathbf{0}, \mathbf{I})$. Then we feed $(z_1, 0)$ to D to produce $\hat{\mu}_1$ and $\hat{\sigma}_1$ and then $x_1 = \hat{\mu}_1 + \hat{\sigma}_1 \nu_1$. Next we feed in (z_2, x_1) to produce $\hat{\mu}_2$ and $\hat{\sigma}_2$ and then $x_2 = \hat{\mu}_2 + \hat{\sigma}_2 \nu_2$, and so on, up to x_t.

In our prototype, both the encoder and the decoder contain a Long Short-Term Memory (LSTM) layer (Hochreiter and Schmidhuber 1997), which allows for concise capture of temporal patterns when encoding the training data. We refer to the resulting neural architecture as NIM-VL. If the x_i's are known *a priori* to be i.i.d. or bounded, we can modify and simplify the architecture to increase accuracy and speed.

The key idea is to jointly (i) train 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 normal samples having these means and variances will jointly be distributed as a sample of the target stochastic process, and (ii) train E so that outputs $z_1, \ldots, z_t$, taken together, look like i.i.d. samples from a standard normal distribution $N(0,1)$, since normal samples are input to D during generation. 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 a loss function that is formulated to achieve goals (i) and (ii). Briefly, one term of the loss function represents the KL-divergence between $N\big(\tilde{\boldsymbol{\mu}}, \mathrm{diag}(\tilde{\boldsymbol{\sigma}})\big)$ and $N(\mathbf{0}, \mathbf{I})$ and one term represents the negative log-likelihood of $\mathbf{x}$ under the $N\big(\hat{\boldsymbol{\mu}}, \mathrm{diag}(\hat{\boldsymbol{\sigma}})\big)$ distribution. (Use of a Gaussian distribution is convenient for backpropagation.) The output of the training step is a GetNextVar() function that gets called repeatedly during a simulation run.

3 EXPERIMENTS

We tested NIM-VL on two complex, nonstationary stochastic processes: a mixture of two nonstationary ARMA processes and the interarrival time sequence for a nonhomogenous Poisson process (NHPP). We also used NIM-VL to learn the arrival and service processes for a NHPP/Gamma/1 FIFO queue and then estimated $\overline{W}_{100}$, the average waiting time for the first 100 customers. In each case, we compare processes and simulations generated the usual way ("ground truth") versus those generated by NIM-VL after training. Our graphical results indicate close agreement: e.g., small absolute differences in $\widehat{\mathrm{Corr}}[X_i, X_j]$ for the ARMA/ARMA mixture (max difference < 0.061), almost identical empirical intensity functions $\hat{\lambda}(t)$ for the NHPP, and almost identical empirical density functions for $\overline{W}_{100}$. Generation speeds were also acceptable: on a commodity 2019 MacBook Pro, NIM, as implemented in PyTorch, is able to generate 1,000 sequences of 1,000 learned NHPP interarrival times in roughly 0.85 seconds and 10^6 i.i.d learned exponential random variates in roughly 0.12 seconds. In conclusion, NIM has the potential to help overcome one of the key barriers to simulation for non-experts.

REFERENCES

Doersch, C. 2016. "Tutorial on Variational Autoencoders". *arXiv preprint arXiv:1606.05908v2*.

Hochreiter, S., and J. Schmidhuber. 1997. "Long Short-Term Memory". *Neural Computation* 9(8):1735–1780.