NIM: Generative Neural Networks for Simulation Input Modeling

Wang Cen, Emily Herbert, Peter Haas

University of Massachusetts Amherst

Simulation Input Modeling

- Simulations are widely used to improve existing systems, e.g.
  - Emergency rooms, call centers, finance, manufacturing, …

- Input models are fitted to represent input processes to a system
  - New samples from the models are drawn to drive simulations
  - Probability distributions: $\exp(\lambda)$, $\mathrm{Beta}(\alpha, \beta)$, etc.
  - Stochastic processes: ARMA, NHPP, etc.

Input Modeling is Key to Simulation

- Faithful input models help ensure credible results

- **But hard!**
  - Distribution-fitting software fits many distribution families on historical data and recommends the best one based on GoF metrics
  - Current software fails for complex i.i.d. distributions and stochastic processes

- **Good news**: increasingly abundant data
  - IoT sensors, logs, annotated machine vision, …

True Distribution	Estimated Distribution	Estimated Goodness of Fit
i.i.d. beta	beta	Good
i.i.d. exp	Gamma	Good
i.i.d. Gaussian mixture	Johnson SU	Bad
i.i.d. Gamma-Uniform	Joshson SB	Bad
ARMA	Johnson SU	Good
NHPP	Pearson Type VI	Good
Call center data	Pearson Type VI	Bad

Results from ExpertFit

NIM: Neural Input Modeling

- NIM is a neural-network-based solution to input modeling that exploits abundant data
  - Automatically fits complex stochastic processes
  - Automatically, efficiently generates sample paths
  - Avoids overfitting
  - Can exploit prior knowledge (bounds, i.i.d. structure, multimodality)
- Novel architecture
  - Variational autoencoder
  - Long Short-Term Memory network to concisely capture temporal dependencies

$$\begin{array}{ccccccccccc} \mathbf{x} & & & & \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}} & & \mathbf{z} & & Z & & & & \hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}} \\ \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_t \end{bmatrix} & \longrightarrow & \boxed{E} & \longrightarrow & \begin{bmatrix} \tilde{\mu}_1 \\ \tilde{\mu}_2 \\ \vdots \\ \tilde{\mu}_t \end{bmatrix}, \begin{bmatrix} \tilde{\sigma}_1 \\ \tilde{\sigma}_2 \\ \vdots \\ \tilde{\sigma}_t \end{bmatrix} & \longrightarrow & \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_t \end{bmatrix} & \longrightarrow & \begin{bmatrix} z_1, 0 \\ z_2, x_1 \\ \vdots \\ z_t, x_{t-1} \end{bmatrix} & \longrightarrow & \boxed{D} & \longrightarrow & \begin{bmatrix} \hat{\mu}_1 \\ \hat{\mu}_2 \\ \vdots \\ \hat{\mu}_t \end{bmatrix}, \begin{bmatrix} \hat{\sigma}_1 \\ \hat{\sigma}_2 \\ \vdots \\ \hat{\sigma}_t \end{bmatrix} \end{array}$$

NIM Inspiration

- By inversion method (for continuous RVs):
  - If $Z \sim N(0,1)$, then $X = G(Z) = F^{-1}(\Phi(Z))$ is distributed according to F

- We can extend the idea to a stochastic process $X = (X_1, \ldots, X_t)$
  - $F(x_1, \ldots, x_t) = F(x_1)F(x_2 \mid x_1) \ldots F(x_t \mid x_1, \ldots, x_{t-1})$
  - $Z_1, \ldots, Z_t \sim N(0,1) \quad X_1 = G_1(Z_1), X_2 = G_2(Z_2|X_1), \ldots, X_t = G_t(Z_t|X_1, X_2, \ldots, X_{t-1})$
  - $G_i(z_i|x_1, \ldots, x_{i-1}) = F_i^{-1}\left(\Phi(z_i)|x_1, \ldots, x_{i-1}\right)$
  - We have thus specified G to transform $Z_1, \ldots, Z_t$ to stochastic process X

- Neural networks can learn complex functions like G from data

Outline

- Neural networks & generative neural networks
- NIM-VM for i.i.d random variables
  - Variational Autoencoder (VAE) + Multilayer Perceptron (MLP)
- NIM-VL for stochastic processes
  - VAE + Long Short-term Memory Network (LSTM)
- Experimental Results
  - Accuracy and Performance
- Future Work

Neural Nets & Generative Neural Nets

- Neural Network
  - A means of doing machine learning, in which a computer learns to some complex function by analyzing training examples.
- Generative Neural Network (GNN)
  - Learns a distribution *P(X)* and generates novel samples from it
- We use a type of GNN called a **Variational Autoencoder** (VAE)
  - Accomplishes generation tasks via an ***encoder*** *E* and a ***decoder*** *D*
  - Use encoder to facilitate training (learns internal representation)
  - Use decoder to draw new samples

Synthetic faces

NIM-VM Decoder

$$z \longrightarrow \boxed{D} \longrightarrow (\hat{\mu}, \hat{\sigma}) \longrightarrow x$$

- Observed (real-valued) datapoint *x* is assumed to generated as follows:
  - Sample a *latent variable* z from some prior distribution $P(z)$
  - Feed z into a function *g* that outputs a *data-generation distribution* $P(x \mid z)$
  - x is a sample from the data-generation distribution
  - For convenience, we take $P(z) = N(0, 1)$ and $P(x \mid z) = N(\hat{\mu}, \hat{\sigma}^2)$
  - Notice that $\hat{\mu} = \hat{\mu}(z), \hat{\sigma} = \hat{\sigma}(z)$ so that $\boldsymbol{x} = \hat{\boldsymbol{\mu}}(\boldsymbol{z}) + \hat{\boldsymbol{\sigma}}(\boldsymbol{z})\boldsymbol{\xi}$ **with** $\boldsymbol{\xi} \sim \boldsymbol{N}(\boldsymbol{0}, \boldsymbol{1})$

- Use decoder *D* to learn the complex *g* function

NIM-VM Encoder

$$x \longrightarrow \boxed{E} \longrightarrow (\tilde{\mu}, \tilde{\sigma}) \longrightarrow z \longrightarrow \boxed{D} \longrightarrow (\hat{\mu}, \hat{\sigma})$$

- Encoder *E*
  - Learns the posterior probability $P(z \mid x)$ of the latent variable that produced *x*
  - $P(z \mid x)$ is complex and expensive to compute via Bayes rule
  - *E* approximates it by a simpler distribution $Q(z \mid x) = N(\tilde{\mu}, \tilde{\sigma}^2)$
  - Notice that $\tilde{\mu} = \tilde{\mu}(x), \tilde{\sigma} = \tilde{\sigma}(x)$

NIM-VM Neural Architecture

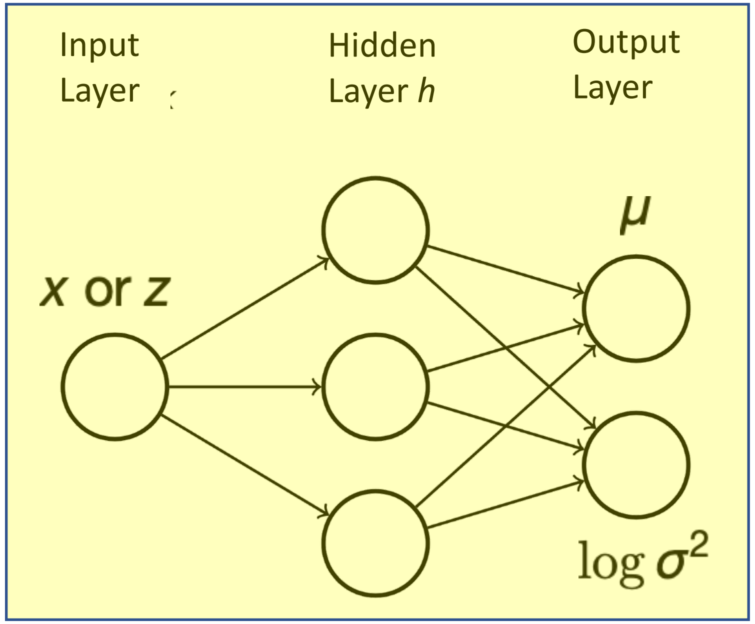
- The encoder and decoder use Multilayer Perceptron (MLP) architecture
  - Encoder *E*:

$$\tilde{h} = \max(0_m, \tilde{W}_1 x + \tilde{b}_1), \quad \tilde{\mu} = \tilde{W}_2 \tilde{h} + \tilde{b}_2, \quad \log \tilde{\sigma}^2 = \tilde{W}_3 \tilde{h} + \tilde{b}_3$$

  - Decoder *D*:

$$\hat{h} = \max(0_m, \hat{W}_1 z + \hat{b}_1), \quad \hat{\mu} = \hat{W}_2 \hat{h} + \hat{b}_2, \quad \log \hat{\sigma}^2 = \hat{W}_3 \hat{h} + \hat{b}_3$$

  - *W*'s are "weights" and *b*'s are "biases", collected in θ
  - θ is learned during training, using data



Training NIM-VM

$$x \longrightarrow \boxed{E} \longrightarrow (\tilde{\mu}, \tilde{\sigma}) \longrightarrow z \longrightarrow \boxed{D} \longrightarrow (\hat{\mu}, \hat{\sigma})$$

- We train NIM-VM by choosing θ to minimize loss function (via SGD)

$$L(x;\theta) = -\frac{1}{2}(\log \tilde{\sigma}^2 - \tilde{\mu}^2 - \tilde{\sigma}^2 + 1) + \frac{1}{2}\left(\log 2\pi + \log \hat{\sigma}^2 + \frac{(x-\hat{\mu})^2}{\hat{\sigma}^2}\right)$$

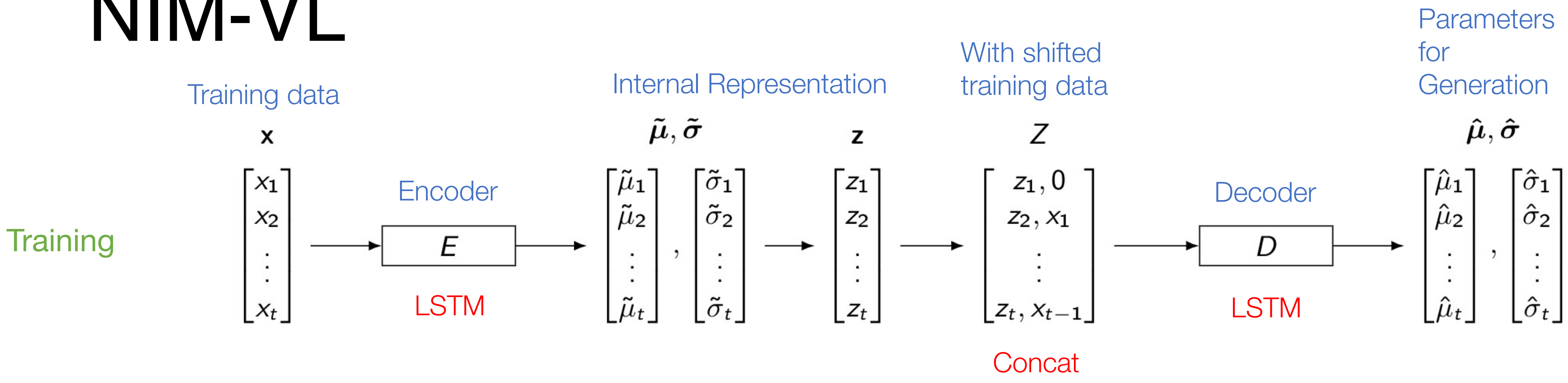
- First term: **KL-divergence** between $Q(z \mid x) = N(\tilde{\mu}, \tilde{\sigma}^2)$ and $P(z) = N(0,1)$
  - z-values produced by the encoder should look like i.i.d. samples from $N(0,1)$
  - Acts as a regularizer, and helps avoid overfitting to data
- Second term: **Reconstruction loss** $E_z[-\log P(x \mid z)]$ where $z \sim N(\hat{\mu}, \hat{\sigma}^2)$
  - The values we sample from $P(x \mid z)$ should look like training data

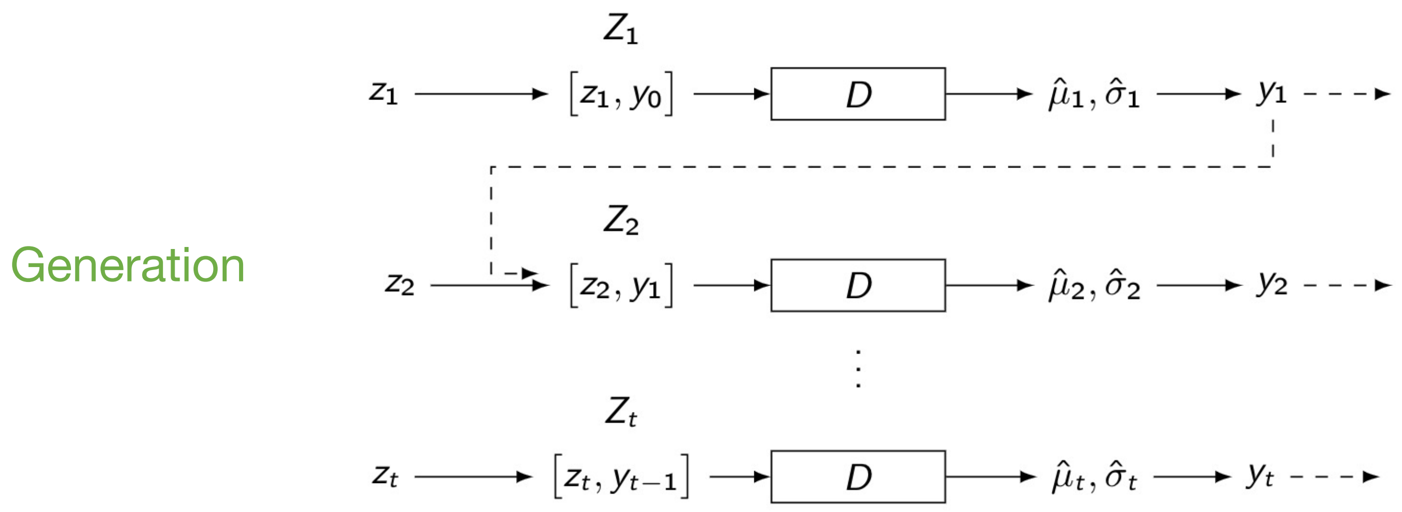
NIM-VM Limitations

NIM-VM works well for i.i.d. random variables (each X is real-valued) but not well for stochastic processes where $X = (X_1, \ldots, X_t)$

- The number of neurons (size of θ) grows linearly with the length of the stochastic process
- NIM-VM can only handle a fixed input and output size
- MLPs are not good at capturing long-range dependencies, key to modeling complex stochastic processes

NIM-VL





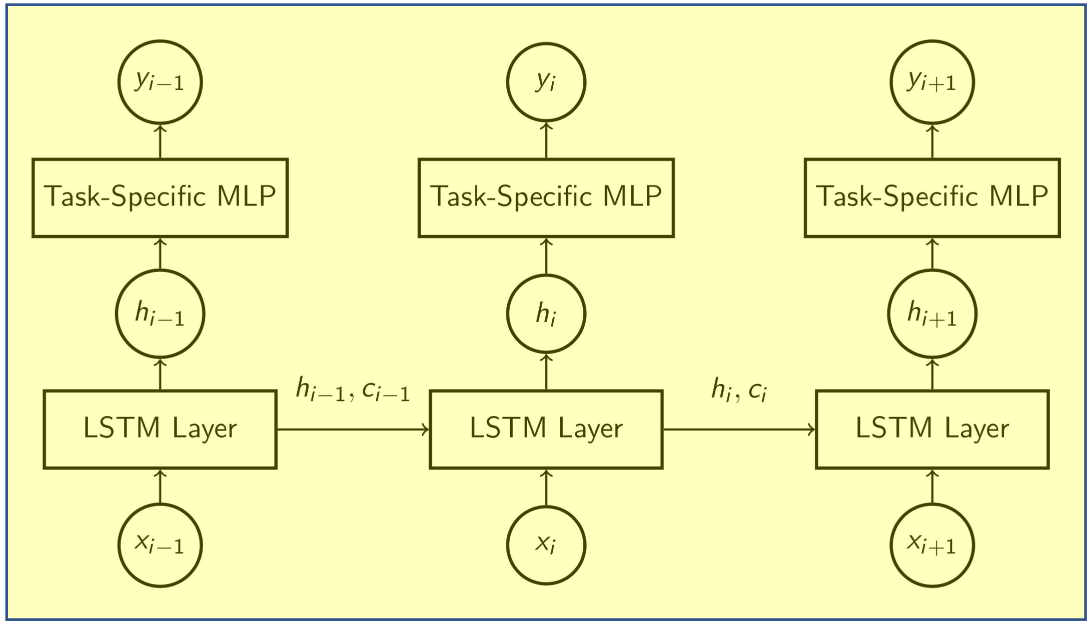
- LSTM to capture long-range dependency
- Concatenation for sequential generation

Long Short-term Memory (LSTM)

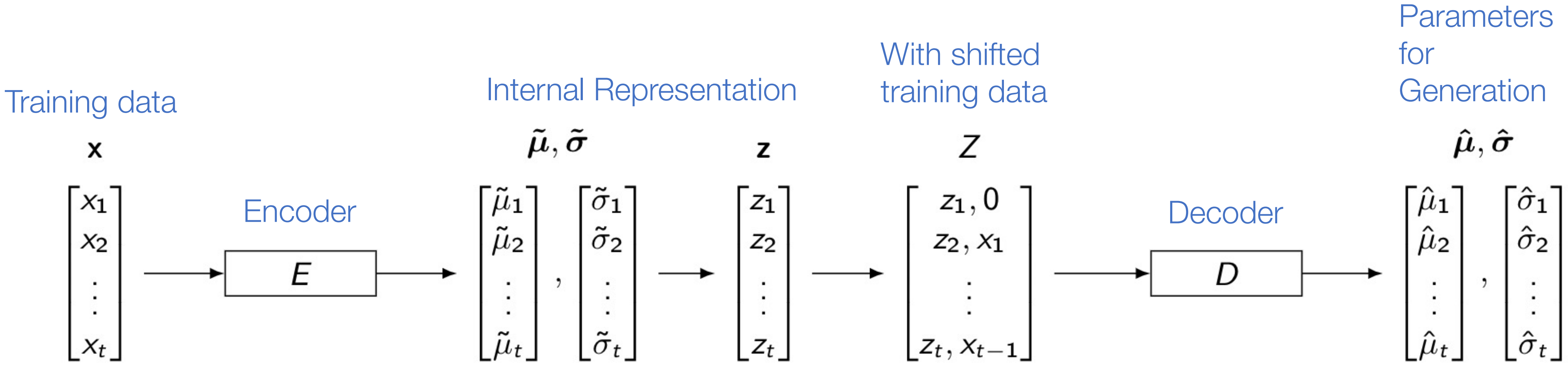
- LSTMs are good at modeling time series
  - They explicitly model temporal dependency across the timesteps
  - At a time step i :

    $$(h_i, c_i) = f_{\mathsf{LSTM}}(h_{i-1}, c_{i-1}, x_i; \theta_{\mathsf{LSTM}})$$

  - h_i : hidden state, c_i : cell state
  - They “remember” what happened in the past



Training NIM-VL



NIM-VM

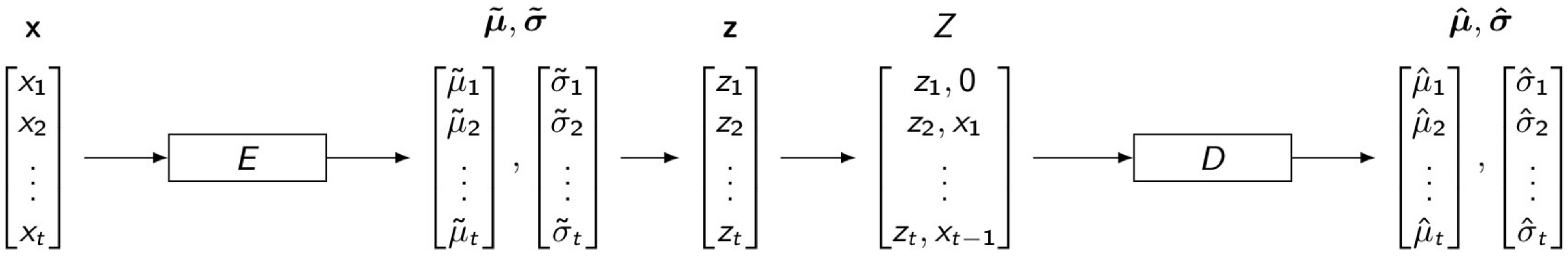
Loss function

$$-\frac{1}{2}(\log \tilde{\sigma}^2 - \tilde{\mu}^2 - \tilde{\sigma}^2 + 1) + \frac{1}{2}\left(\log 2\pi + \log \hat{\sigma}^2 + \frac{(x - \hat{\mu})^2}{\hat{\sigma}^2}\right)$$

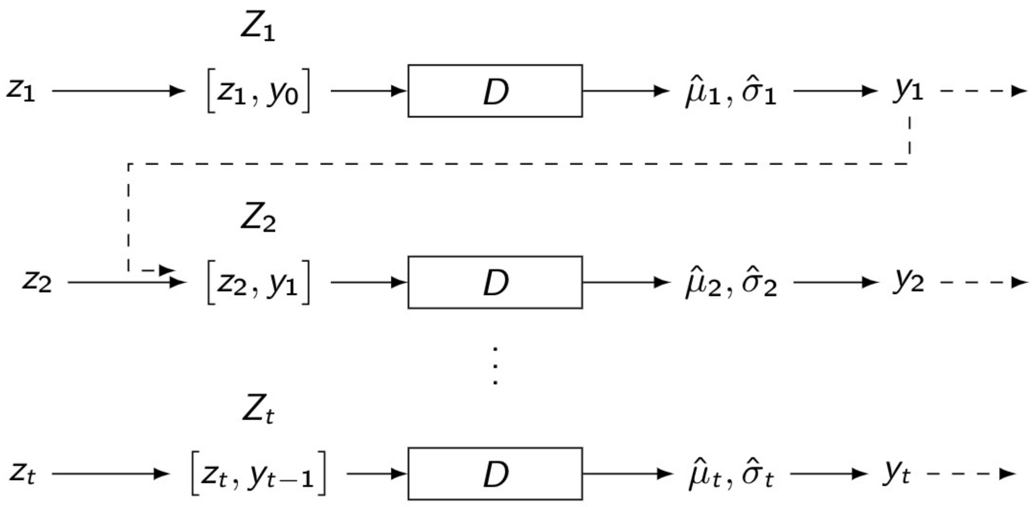
NIM-VL

$$-\frac{1}{2}\sum_{i=1}^{t}(\log \tilde{\sigma}_i^2 - \tilde{\mu}_i^2 - \tilde{\sigma}_i^2 + 1) + \frac{1}{2}\sum_{i=1}^{t}\left(\log 2\pi + \log \hat{\sigma}_i^2 + \frac{(x_i - \hat{\mu}_i)^2}{\hat{\sigma}_i^2}\right)$$

NIM-VL Generation



1. Sample $z_i \sim N(0,1)$
2. Compute $\hat{\mu}_i$ and $\hat{\sigma}_i$
3. Sample $y_i \sim N(\hat{\mu}_i, \hat{\sigma}_i^2)$
4. $i \leftarrow i+1$ and repeat



Exploiting Domain Knowledge*

- I.i.d. property: Use NIM-VM
  - Simpler and faster than NIM-VL
  - Won't spuriously estimate (nonexistent) dependencies
- Bounded random variables: Use transformations
  - Apply nonlinear transformation to map each training x to real line
  - Apply inverse transformation to NIM generated output
- Multimodal distributions: Mixture models
  - Replace $N(\hat{\mu}, \hat{\sigma}^2)$ in the decoder by a Gaussian mixture model

*See paper for details

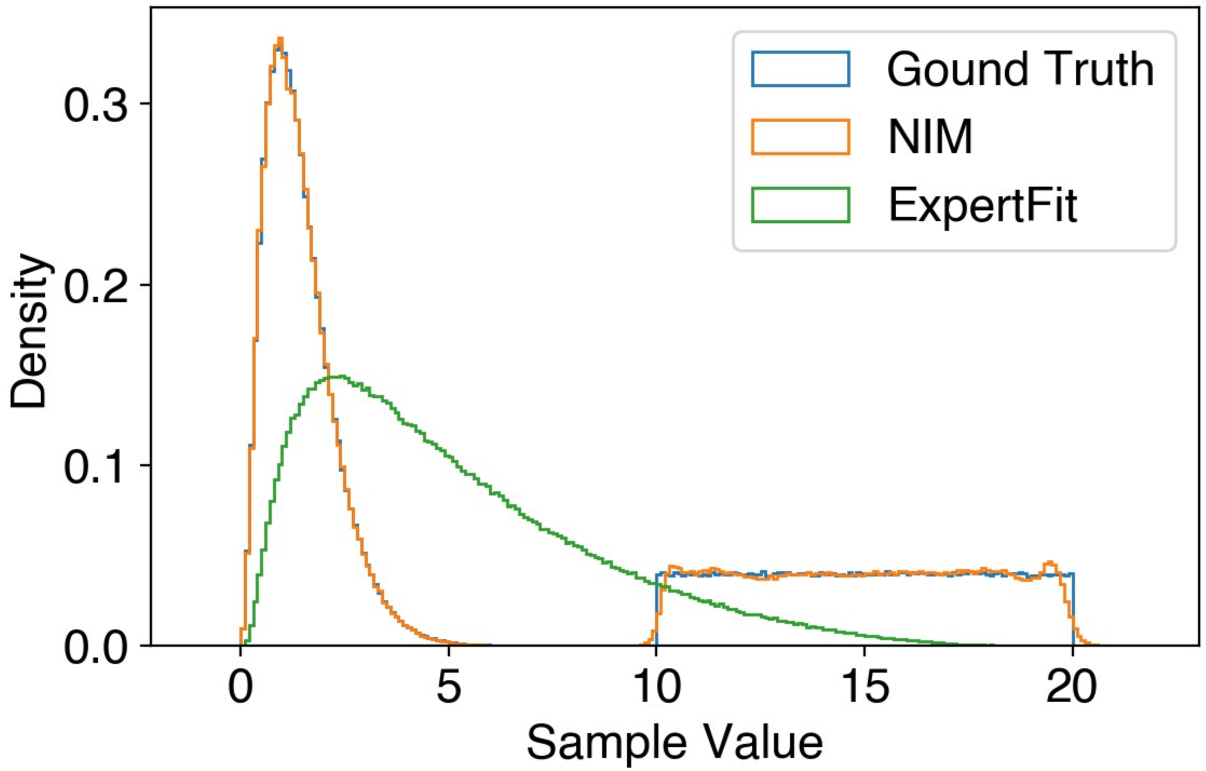
Some Experimental Results

- See paper for more experiments and details

Accuracy: I.I.D. Multimodal Distribution

- 100,000 training samples
- Compare empirical densities
  - Ground truth (exact density)
  - NIM-VL with Gaussian mixture
  - ExpertFit

i.i.d. Gamma Uniform mixture



0.6 * Gamma(2.875, 0.5) + 0.4 * Uniform(10,20)

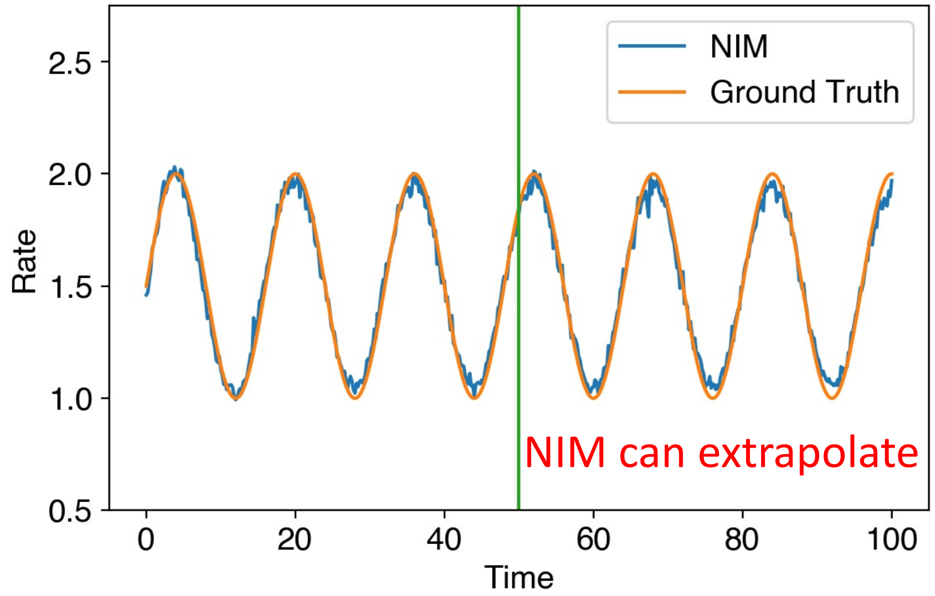
Accuracy: Complex Stochastic Process

- Non-homogeneous Poisson Process

- Trained NIM-VL
  - 1,000 sample paths on [0,50]
  - Used log-transformation since interarrival times are positive

- Compared empirical arrival rates to ground truth

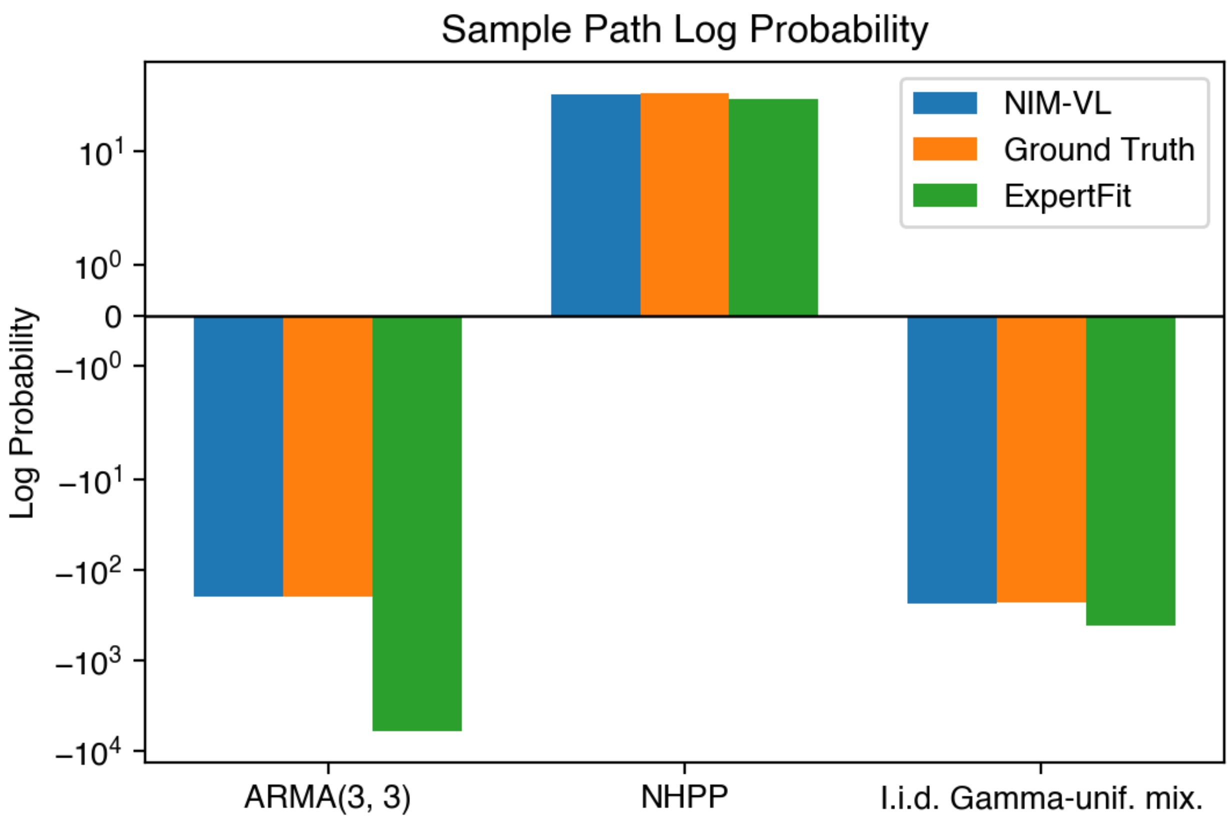
Non-homogenous Poisson Process - Arrival Rate



$$\lambda(t) = \frac{1}{2}\sin(\frac{\pi}{8}t) + \frac{3}{2}$$

Accuracy: Mean Log-Likelihood

- Compute mean log-likelihood over 1000 sample paths
  - From ground truth distributions
  - From distributions learned by NIM
  - From distributions learned by ExpertFit
- Log probability of NIM sample paths and ground-truth sample paths are close
- Larger values than ExpertFit



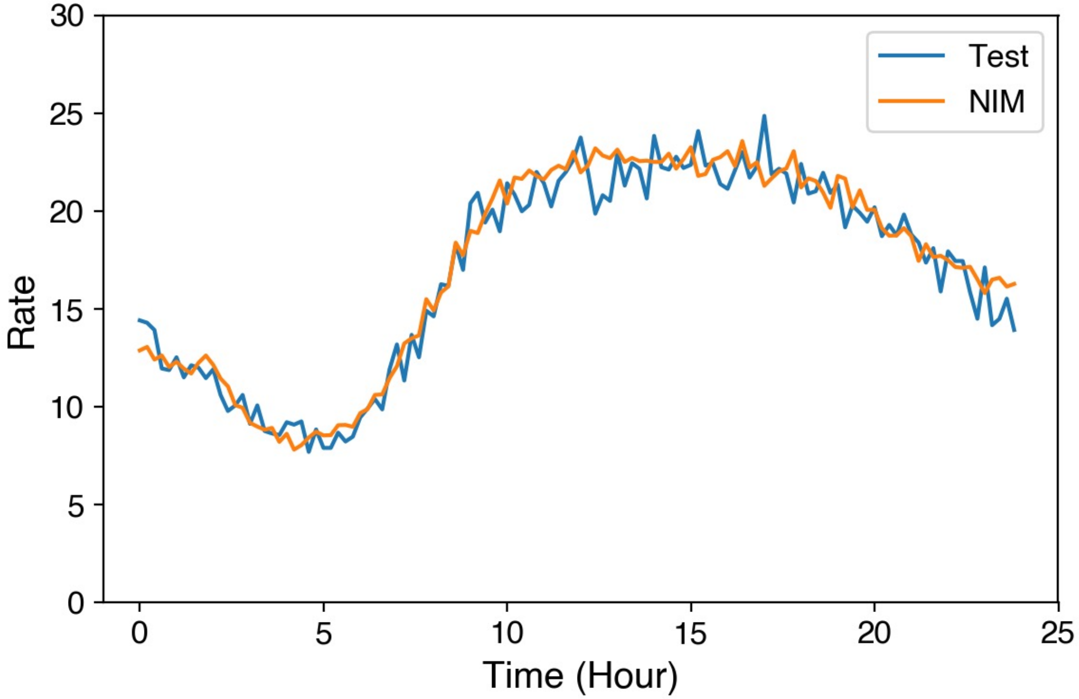
Accuracy: Real-World Data

- San Francisco Fire Department call center: Call interarrival times

- One year of data
  - 2/3 used for training (243 days)
  - 1/3 used for test (122 days)

- Compare empirical call-arrival rates

Empirical Call-Arrival Rate



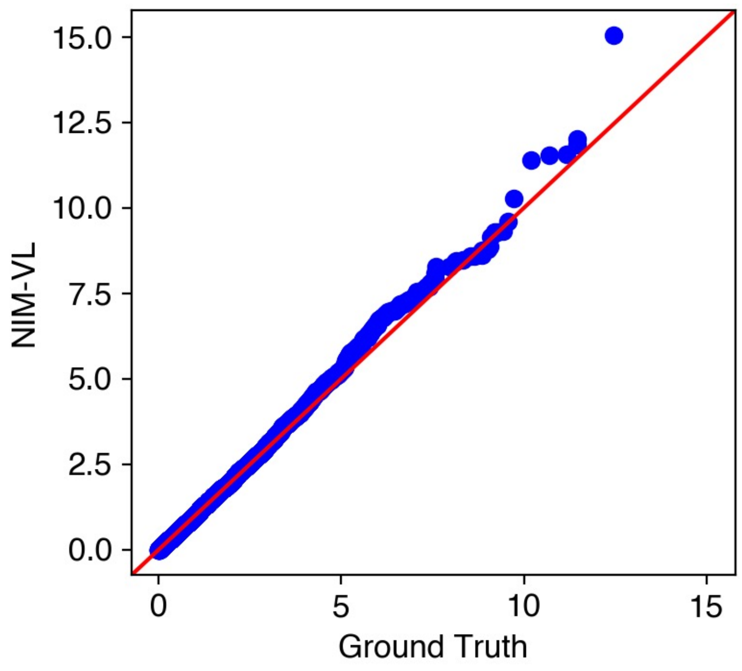
Accuracy: End-to-End Simulation

- Single-server FIFO queue
- Arrival process is NHPP

$$\lambda(t) = \frac{1}{2}\sin\left(\frac{\pi}{8}t\right) + \frac{3}{2}$$

- Processing times are i.i.d. Gamma(1.2, 0.4)
- Simulate waiting time for 60^{th} job
  - Using ground-truth input distributions
  - Using input distributions learned by NIM-VL (1000 sample paths for training)

Q-Q Plot: Dist'n of 60^{th} Waiting Time



Performance

- **Training times**
  - On workstation with 2.10 GHz Intel CPU + NVIDIA GPU
  - Training times between 10-20 minutes
- **Generation times**
  - On a commodity 2018 MacBook Pro
  - 1 million i.i.d. learned exponential random variables in 0.12 seconds
  - 1,000 sequences of 1,000 learned NHPP interarrival times in 0.85 seconds
  - Basically matrix multiplications: Can be further improved using GPU
- **Training-set size**
  - What is smallest training set size to get results comparable to 1,000 training sample paths?
  - ARMA(3,3): 10   NHPP: 250   Gamma-unif mixture: 1,000
  - The simpler the distribution, the less training data is needed

Conclusion and Future Work

- Generative NNs are a promising tool for simulation input modeling in abundant-data scenarios
  - Automated
  - Minimal assumptions
  - Can capture complex statistical structure
- Future work
  - Conditional NIM for what-if analysis and transferring models
  - Nonstationary processes
  - Discrete random inputs
  - Marked point processes
  - Multidimensional processes

Horses with hats

Thanks!

Source code available at: https://github.com/cenwangumass/nim