Tutorial: Artificial Neural Networks for Discrete-event Simulation

Peter J. Haas

Manning College of Information and Computer Sciences

University of Massachusetts Amherst

Winter Simulation Conference

December 17, 2024

MACHINE LEARNING & SIMULATION: FRIENDS OR FOES?

Machine Learning:

- Statistical model to produce predictions or generate data
- Leverages large amount of available data

Adobe Firefly

Simulation:

- Mechanistic model of system logic to produce predictions and generate data
- Incorporates deep domain expertise (logistics, engineering, healthcare, telecom, computer design, …)

Opportunities to achieve the best of both worlds!

- Simulation for ML: E.g., generate training data
- ML for simulation:
  - “Classic” ML for simulation (random forests, SVMs, …)
  - Causal probabilistic graphical models for simulation metamodeling
  - Artificial Neural Networks (ANNs) for simulation: This tutorial

OUTLINE

- Background on ML and ANNs
- ANNs for simulation input modeling
- ANNs for simulation metamodeling and optimization
- Other applications of ANNs to simulation
  - Modeling of agent behavior
  - Simulation validation
  - Variance reduction

CAVEATS

- This is a **tutorial**, not a survey or literature review
  – It strongly reflects my personal experience

- It is a **snapshot** from long ago (June 2024) referencing prehistoric times (2019-20)
  – ANN technology has been developing VERY fast (1 human year = 50 ML years)

- I will not discuss foundation models (e.g., LLMs) very much
  – Will focus on ANNs for stochastic processes (more modest data requirements)

OUTLINE

- Background on ML and ANNs

- ANNs for simulation input modeling

- ANNs for simulation metamodeling and optimization

- Other applications of ANNs to simulation
  - Modeling of agent behavior
  - Simulation validation
  - Variance reduction

PREDICTIVE ML BASICS

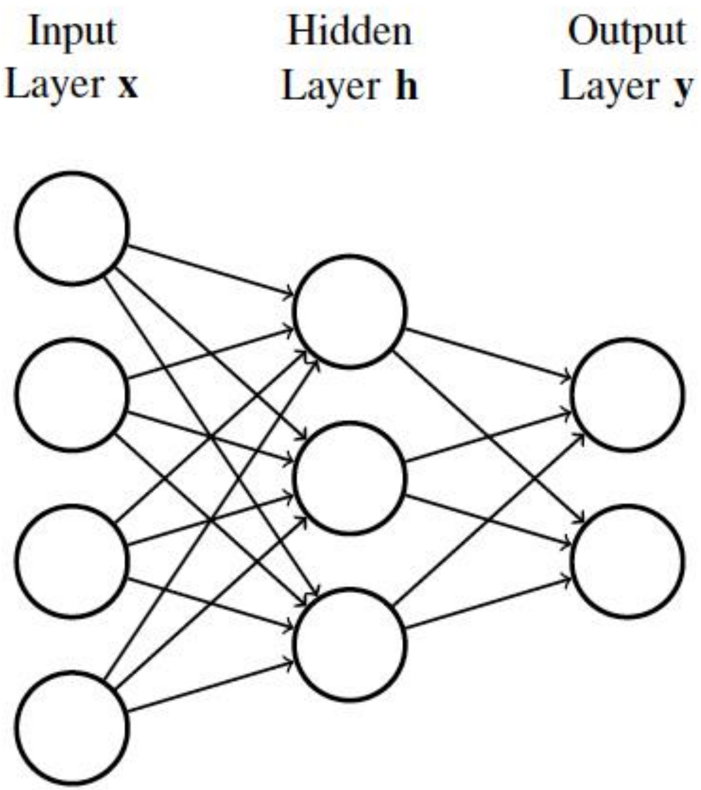
- The simplest setup: Learn a function $f(x;\theta): \Re^d \rightarrow \Re$ given training data $\mathcal{T} = \{(x_i, y_i)\}$
  - Training points are i.i.d. samples from underlying joint probability distribution $P(X, Y)$
  - x_i is called a feature vector
  - θ is a vector of function parameters

- The function is trained (fit to data) by minimizing an (approximated) loss function $\ell(\theta)$
  - E.g., $\ell(\theta) = E_P[(f(X;\theta) - Y)^2]$ is approximated by $|\mathcal{T}|^{-1} \sum f(x_i;\theta) - y_i)^2$
  - This procedure is called empirical risk minimization

- Example: Classical linear regression
  - $f(x;\theta) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$
  - Under mean-squared loss, $\theta^* = (\mathbf{X}^{\mathrm{t}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{t}}\mathbf{y}$ where ith row of $\mathbf{X}$ is $(1, x_{i,1}, \ldots, x_{i,d})$

MULTI-LAYER PERCEPTRON (MLP): REGRESSION ON STEROIDS

- The simplest ANN



Weights and biases

$$h = \sigma(W\mathbf{x} + \mathbf{b})$$

$$\sigma(x) = \mathrm{ReLU}(x) = \max(0, x)$$

- Universal approximation theorems: MLP can approximate any continuous function
  - With single, wide-enough hidden layer [Hornik '91]
  - With enough fixed-width hidden layers (overall fewer neurons) [Hanin & Selke '17]

AUTOMATIC DIFFERENTIATION FOR TRAINING MLP

- Train via gradient descent to minimize loss
- Efficiently compute gradients via autodiff
  - Advantages: fast & more accurate than finite difference
  - Out-of-box in PyTorch, TensorFlow, Jax
- Break functions into sequence of elementary operations
  - Matrix multiplications, nonlinear functions, etc.
  - Cache intermediate results in **forward pass** that computes loss *L*
  - Use chain rule to compute gradients during **backward pass**

Weights

$$\theta = (\theta_1, \theta_2, \theta_3)$$

Data

$$(x, x^2, y)$$

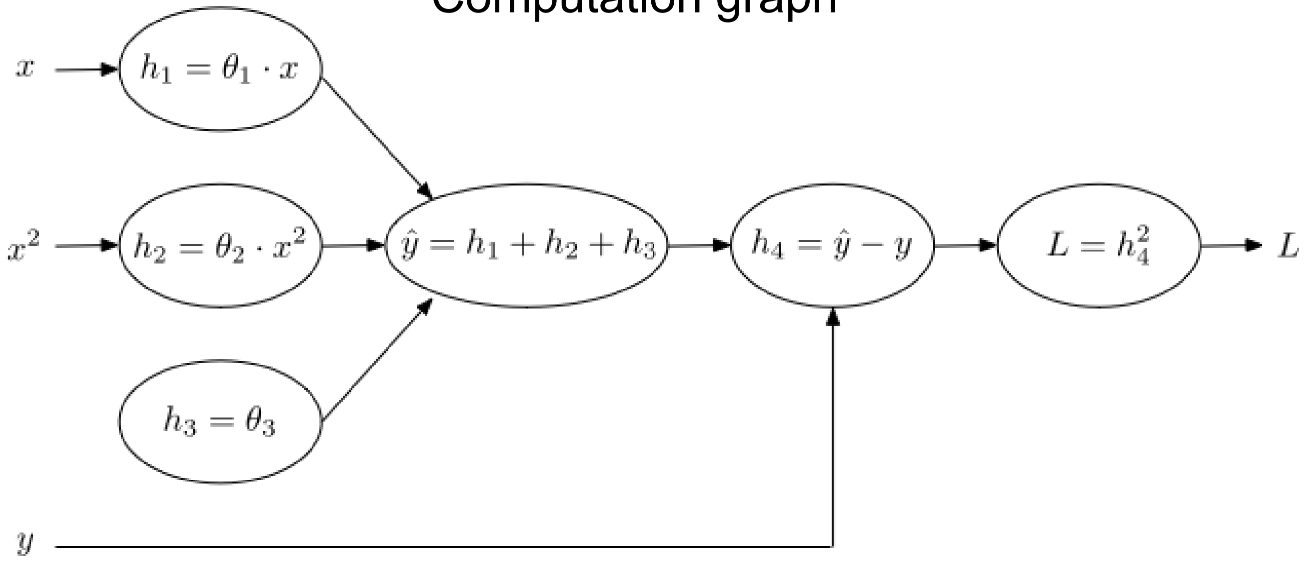
Model

$$y = \theta_1 x + \theta_2 x^2 + \theta_3$$

Loss

$$L = (\tilde{\theta}_1 x + \tilde{\theta}_2 x^2 + \tilde{\theta}_3 - y)^2$$

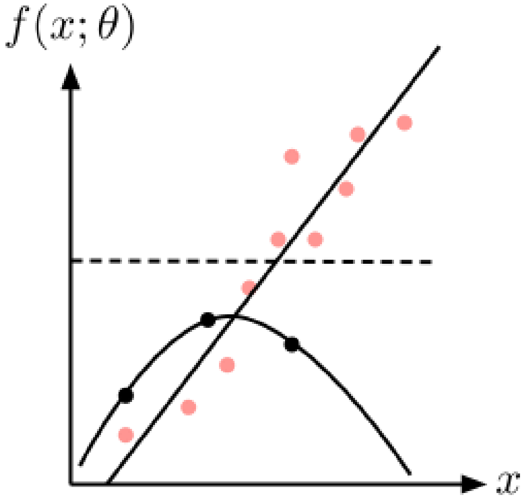
Computation graph



Gradient

$$\frac{\partial L}{\partial \theta_2} = \frac{\partial L}{\partial h_4} \frac{\partial h_4}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial \theta_2} = 2h_4 x^2$$

OVERFITTING AND GENERALIZATION



- Overfitting: ML model may too closely fit training data, yielding large test errors
- ANNs have huge numbers of parameters□is this a problem?

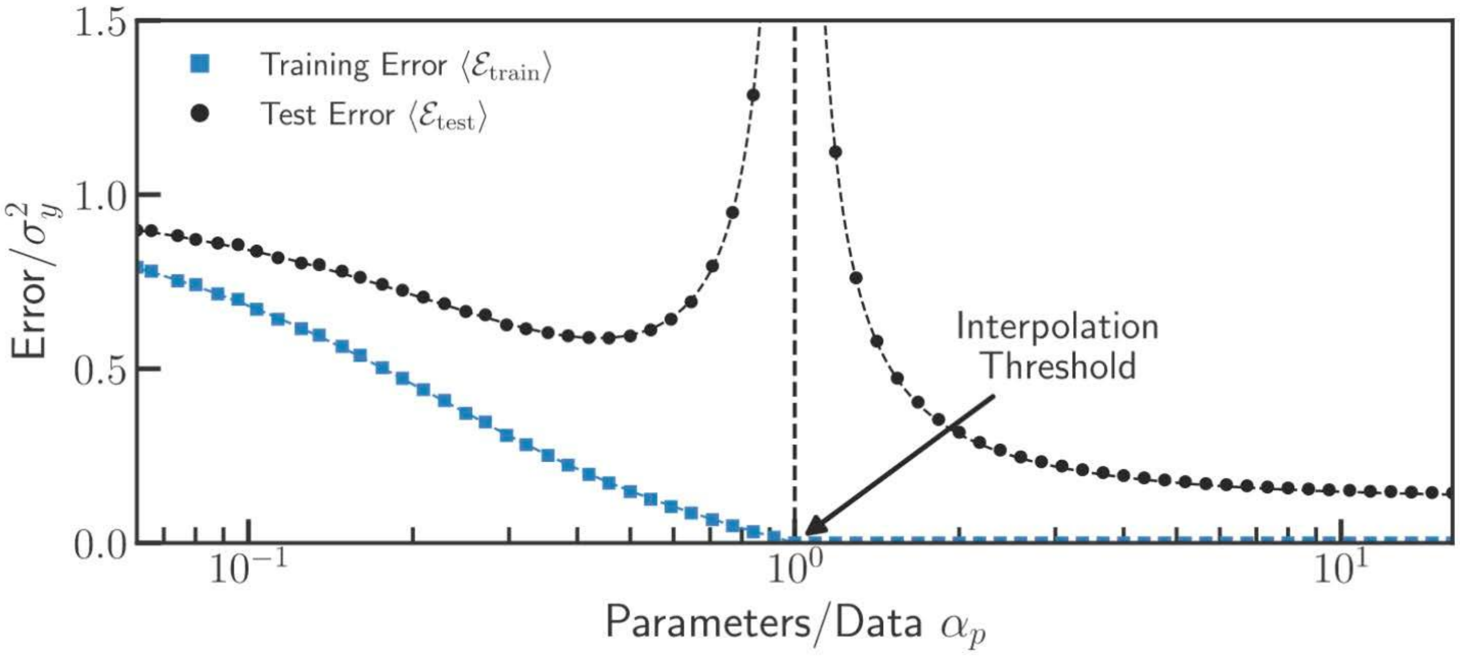
Often for very large models the answer is no!

(“Double descent” behavior)

[Jacot et al. ‘18; Lee et al. 2019]

[Schaffer et al. ’24]

Still need to be careful for moderate-size models!



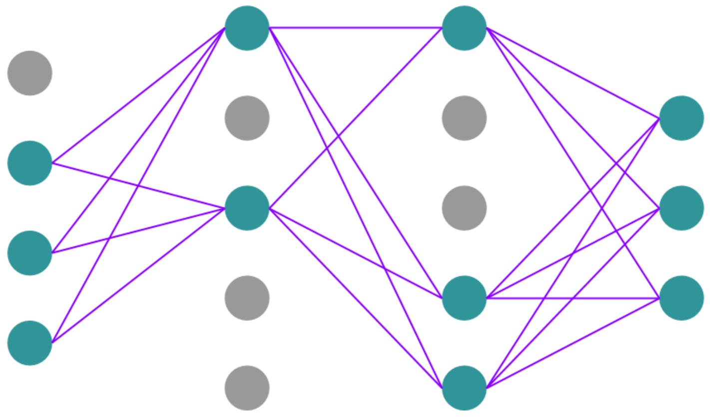
THREE STRATEGIES FOR AVOIDING OVERFITTING

- autoML
  - Divide ground-truth data into a training set and validation set
  - Fit ML model using training set and measure loss on validation set
  - If bad test results, modify model (more layers, smaller GD step-size, etc.) and try again

- Regularization
  - Add term to loss function that penalizes for too many (or high-valued) parameters, e.g

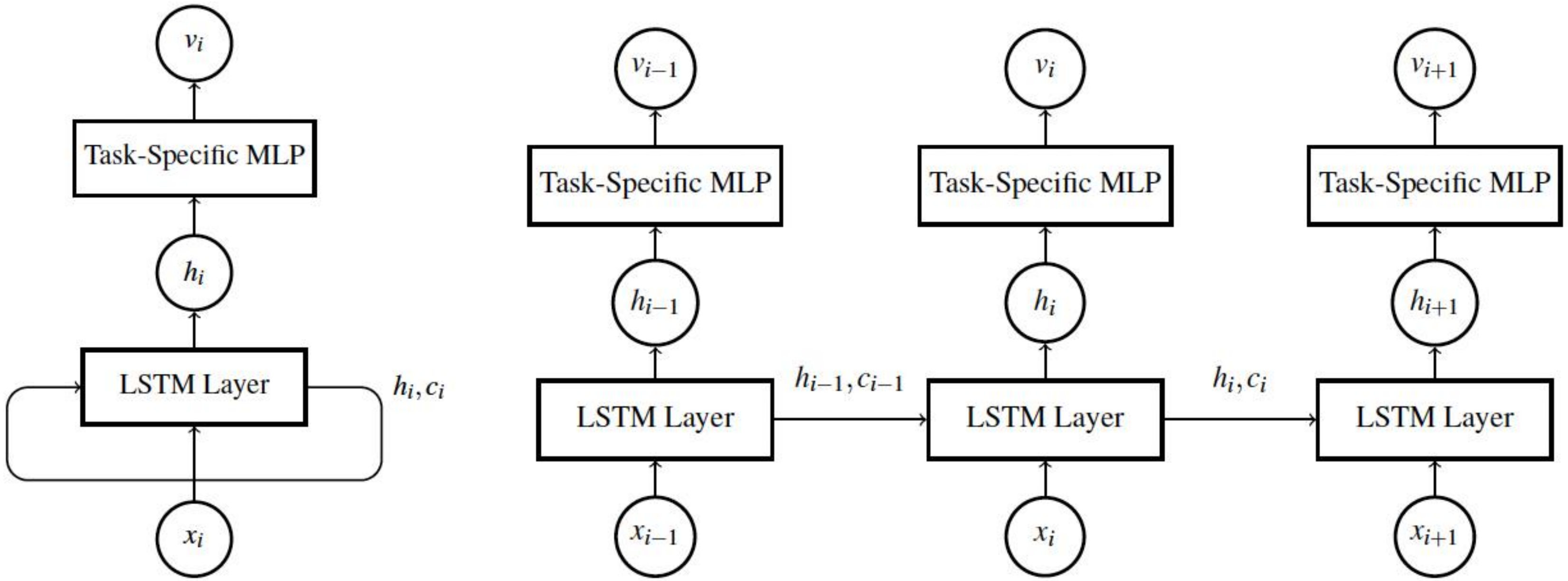
$$\ell'(\theta) = \ell(\theta) + \sum_i |\theta_i| \quad \text{(Lasso regression)}$$

- Drop-out: During each forward pass, independently set output of each neuron to 0 with prob. *p*
  - Equivalent to a form of regularization [Hinton et al. '12]
  - Can use to roughly estimate uncertainty of a trained model (reminiscent of bootstrap, but need, e.g. conformal prediction to get true confidence intervals)



RECURRENT NEURAL NETWORKS: LSTM'S

- Recurrent neural networks (RNNs): Neuron output can feed back into network
- Ex: Long Short-Term Memory (LSTM) components
  - Designed for learning from time-series data
  - Not too many neurons [as in an MLP attempt with features $x = (x_1, \ldots, x_t)$]
  - Can predict beyond training-sample path length
  - Can capture long-range dependencies [Lipton '15]



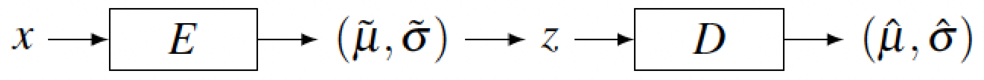
(a) LSTM (b) Unrolled LSTM

GENERATIVE NEURAL NETWORKS: VAE'S

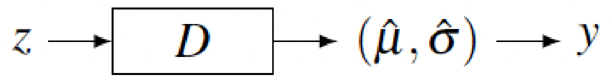
- Goal of GNN: Learn underlying dist'n $P(X)$ from i.i.d. samples of X then generate from $P(X)$

- Variational autoencoders (VAEs)
  - Generative model for observed data:
    1. Sample from latent dist'n [$N(0,1)$]
    2. Feed into function that generates data-generation dist'n
    3. Sample from data-generation dist'n

  - Data-generation distribution [decoder D]: $P(y|z) = N(\hat{\mu}(z), \hat{\sigma}(z))$
  - Latent-space mapping [Encoder E]: $P(z|x) \approx Q(z|x) = \mathrm{N}(\tilde{\mu}(x), \tilde{\sigma}(x))$
  - Loss function tries to ensure:
    - $\mathrm{N}(\tilde{\mu}(x), \tilde{\sigma}(x))$ samples together look like samples from $N(0,1)$ (acts as a regularizer term)
    - $N(\hat{\mu}(z), \hat{\sigma}(z))$ samples together look like samples from $P(X)$



(a) Training



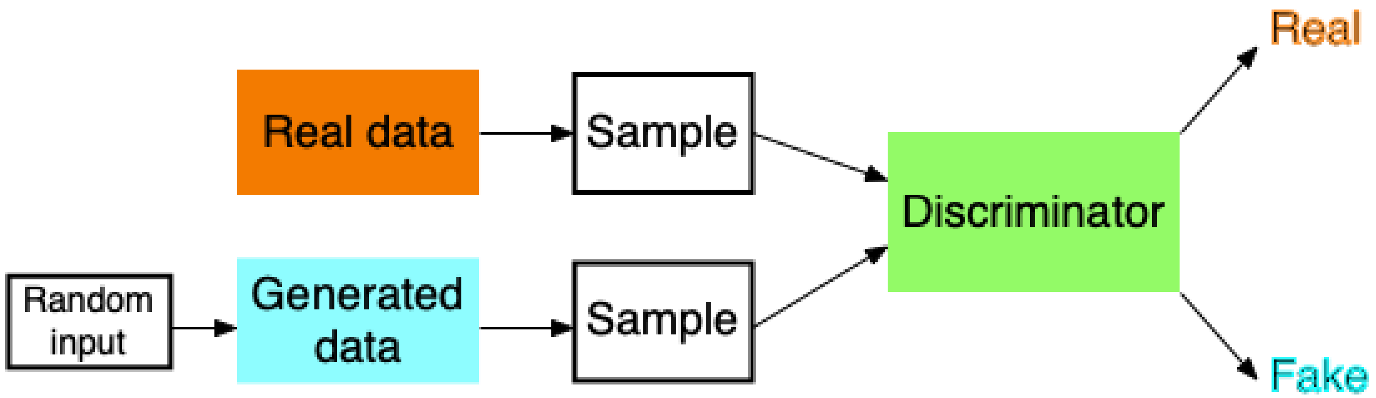
(b) Generation

GENERATIVE NEURAL NETWORKS: GAN'S

- Generative adversarial networks (GANs)
  - Generator tries to generate data that looks like real data
  - Discriminator tries to classify data as real ($D(x;\theta_D) \approx 1$) or generated ($D(x;\theta_D) \approx 0$)
  - Objective function represents misclassification by Discriminator (minimax game):

$$\ell(\theta) = -\frac{1}{n}\sum_{x\in R} \ln\big(D(x;\theta_D)\big) - \frac{1}{n}\sum_{x\in G} \ln(1 - D(x;\theta_D))$$

  - Optimal Generator minimizes Jensen-Shannon dist. between real & generated dist'n



- Original loss function was unstable
  - Directly minimize Wasserstein distance (WGAN) [Arjovsky+ '17] $W_1(\mu_1, \mu_2) = \int_{\Re} |F_1(x) - F_2(x)| dx$
  - Recent modifications of WGAN use Wasserstein variants [Mahdian+ '17, Birrell+ '22]

OUTLINE

- Background on ML and ANNs
- ANNs for simulation input modeling
- ANNs for simulation metamodeling and optimization
- Other applications of ANNs to simulation
  - Modeling of agent behavior
  - Simulation validation
  - Variance reduction

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
  - Hand-crafted generation methods needed

- **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	Johnson 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

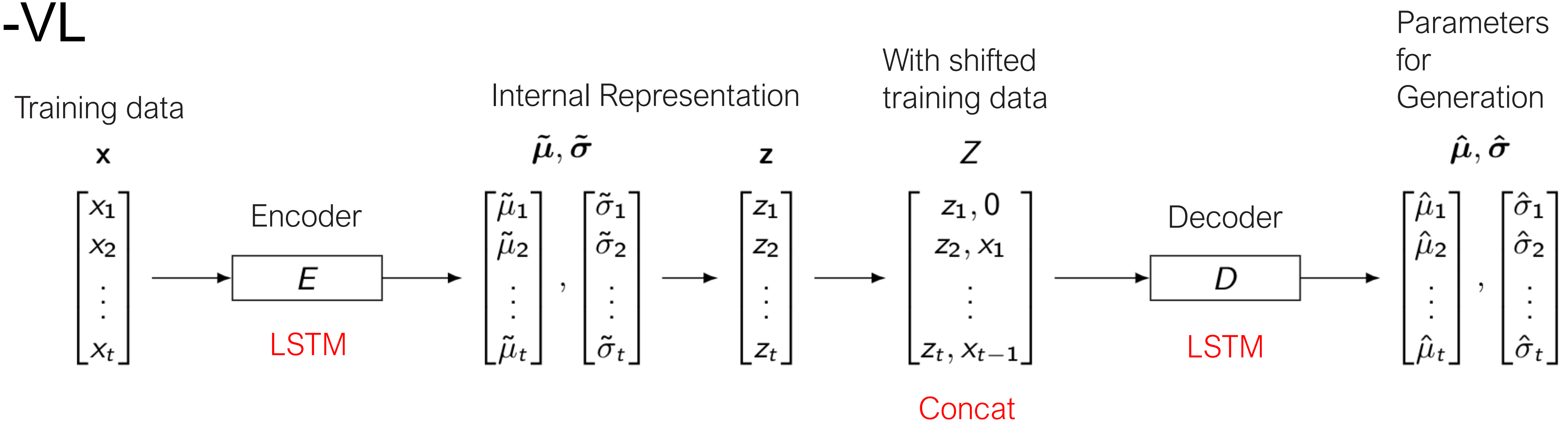
- NIM is a neural-network-based solution to input modeling that exploits abundant data
  - Automatically fits complex stochastic processes
  - Automatically, efficiently generates sample paths
  - Avoids overfitting
  - Can exploit prior knowledge (bounds, i.i.d. structure, multimodality)

- Architecture combines VAE and LSTM

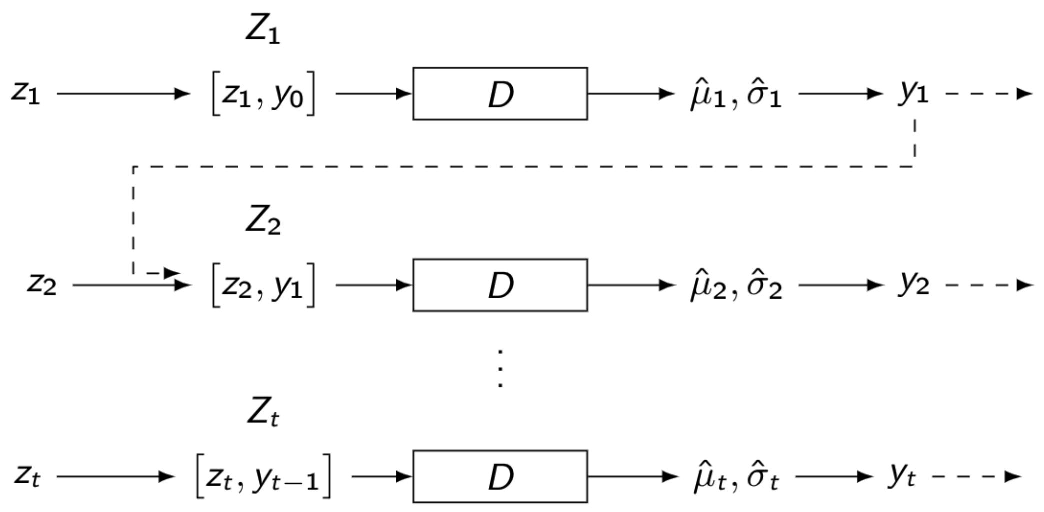
- Motivation: Inversion method
  - If $Z \sim N(0,1)$ then $G(Z) = F^{-1}(\Phi(Z))$ has distribution F
  - Using conditional distributions, can specify G that transforms $Z_1, \ldots, Z_t$ to $X_1, \ldots, X_t$
  - Neural networks can learn complex functions like G from data

NIM-VL

Training

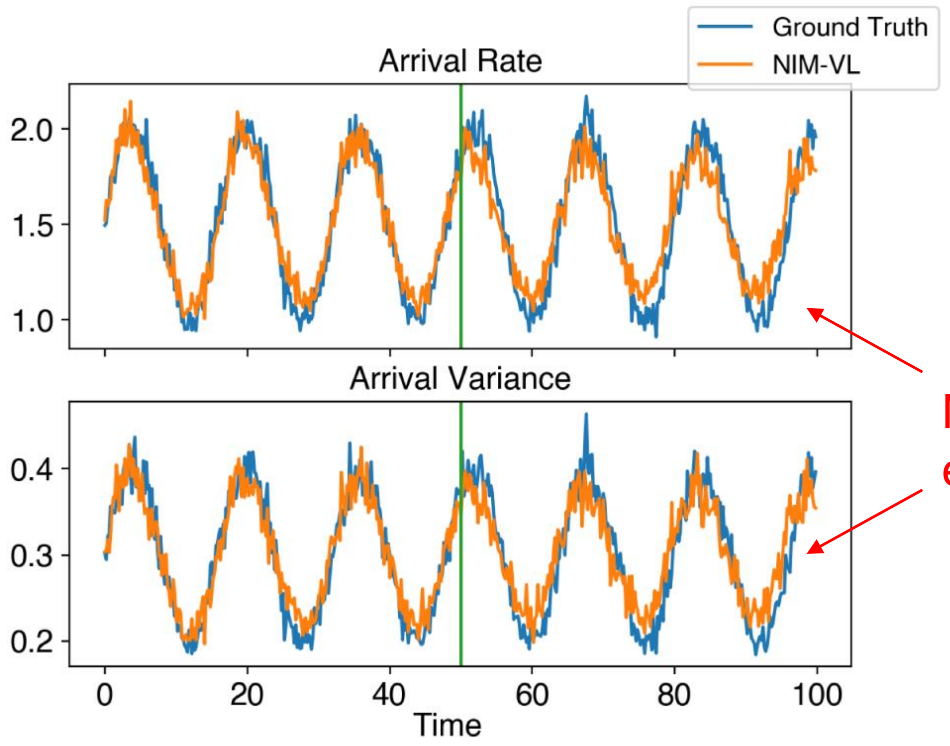


Generation

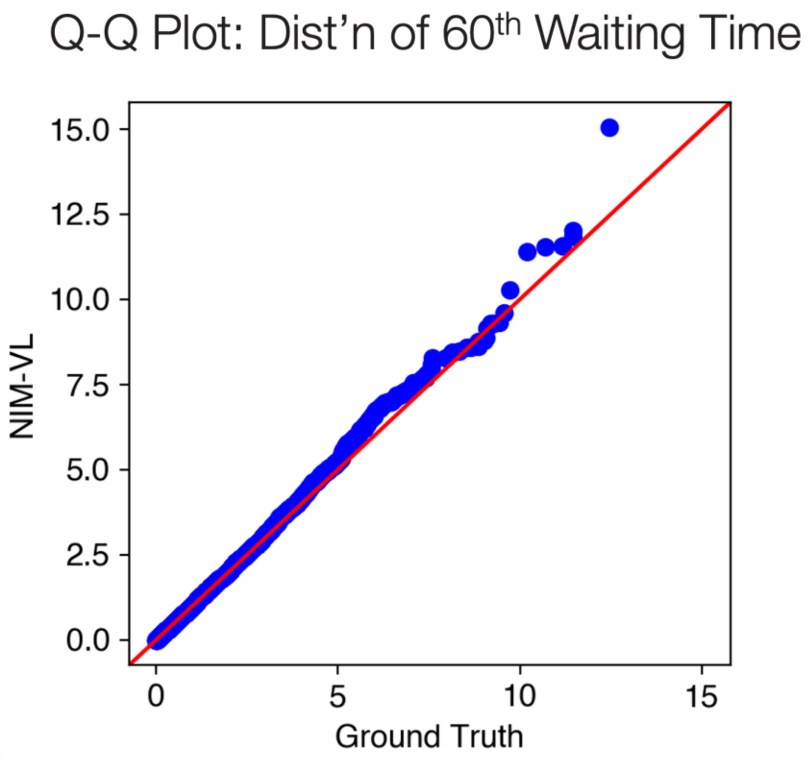


- Use LSTM to capture long-range dependency
- Use concatenation for sequential generation
- Extends to multivariate sequences

EXAMPLES: COMPLEX STOCHASTIC PROCESS



NIM can extrapolate



Non-homogeneous Poisson Process

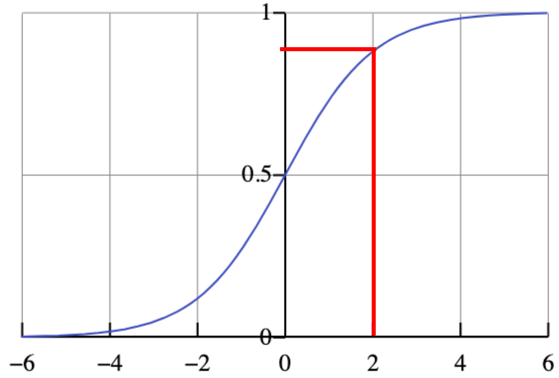
$$\lambda(t) = \frac{1}{2}\sin\left(\frac{\pi}{8}t\right) + \frac{3}{2}$$

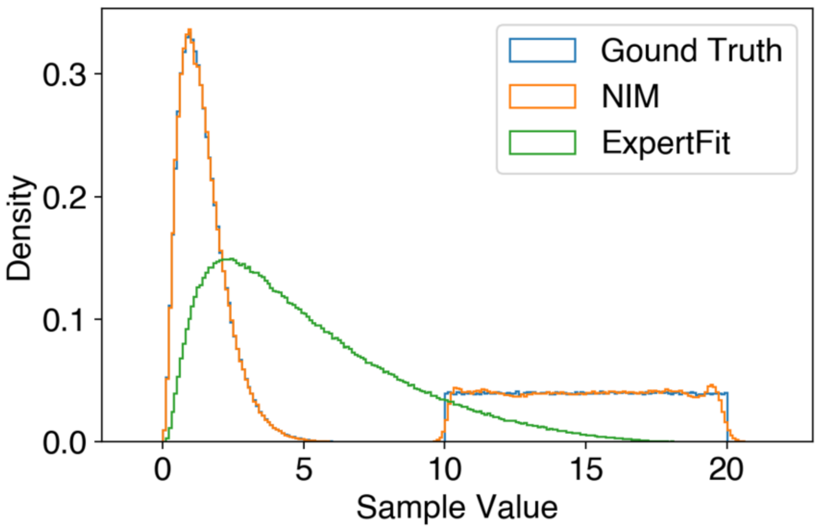
Single-server FIFO Queue

NHPP arrivals, i.i.d. Gamma service times

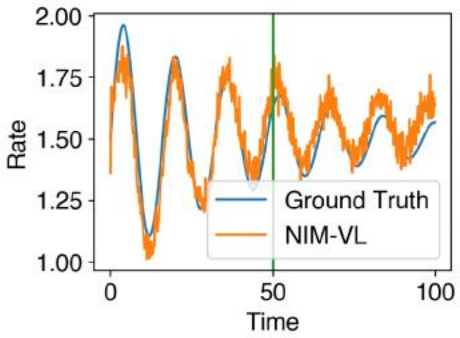
EXPLOITING DOMAIN KNOWLEDGE

- I.i.d. structure: Replace LSTM with MLP
  - Faster, and won't learn spurious autocorrelations
- 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: Gaussian mixture decoder
- Discrete distributions: Softmax decoder $P(v_i) = e^{v_i} / \sum_j e^{v_j}$
- Nonstationary processes: ARIMA-like differencing

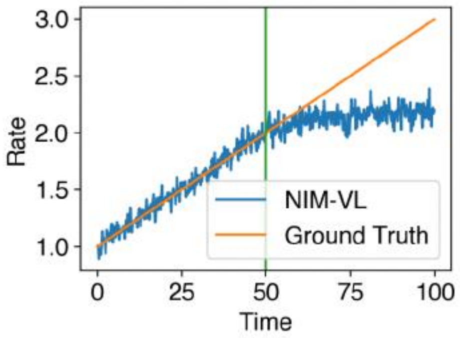




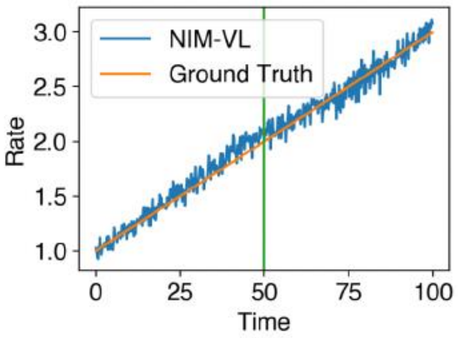
0.6 * Gamma(2.875, 0.5) + 0.4 * Uniform(10,20)



NHPP: Damped sine wave



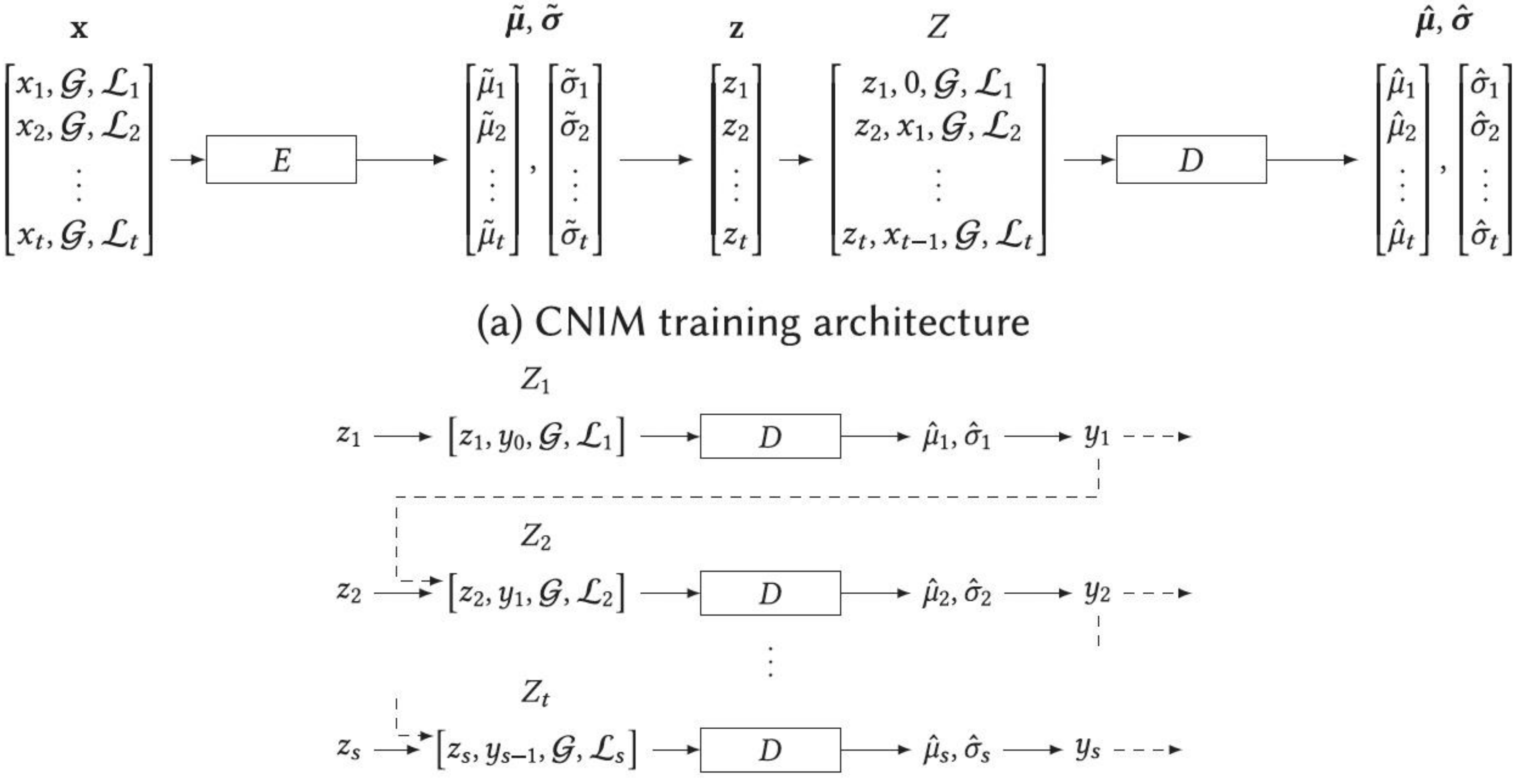
NHPP: No differencing



NHPP: Differencing

EXPLOITING DOMAIN KNOWLEDGE: CONDITIONAL NIM

- Generate sample paths given global or local “condition” (aka context)
  - E.g., arrivals at ice cream stand given daily (global) or hourly (local) temperatures



(a) CNIM training architecture

(b) CNIM generation architecture

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-uniform mixture: 1,000
  - The simpler the distribution, the less training data is needed

OTHER INPUT MODELING TECHNIQUES

- Standard GANs for modeling i.i.d. univariate and bivariate standard distn's [Montevechi+ '21]
- WGANs for modeling doubly stochastic Poisson processes [Zheng & Zheng '21]
- WGANs + recursive model: $X_{k+1} = \mu(l_k, X_k) + \Sigma(l_k, X_k)\eta_{k+1}$ [Zhu+ '23]

OUTLINE

- Background on ML and ANNs
- ANNs for simulation input modeling
- ANNs for simulation metamodeling and optimization
- Other applications of ANNs to simulation
  - Modeling of agent behavior
  - Simulation validation
  - Variance reduction

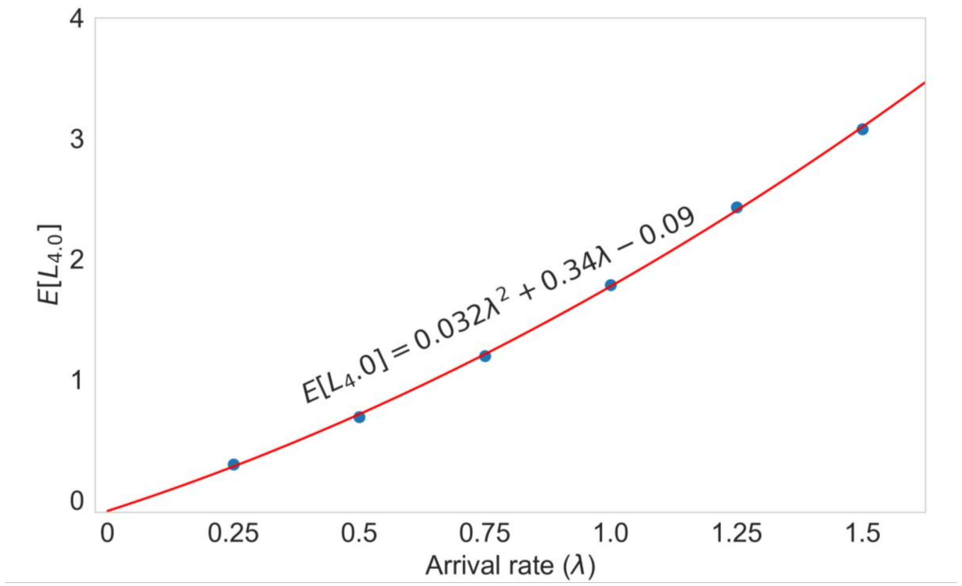
SIMULATION METAMODELING

Why metamodeling?

- Stochastic simulation models of large and complex systems can be *very expensive* to run
  - *Limits use* in tactical or near-real-time settings
  - *Severely limits use* in simulation-based optimization for system design

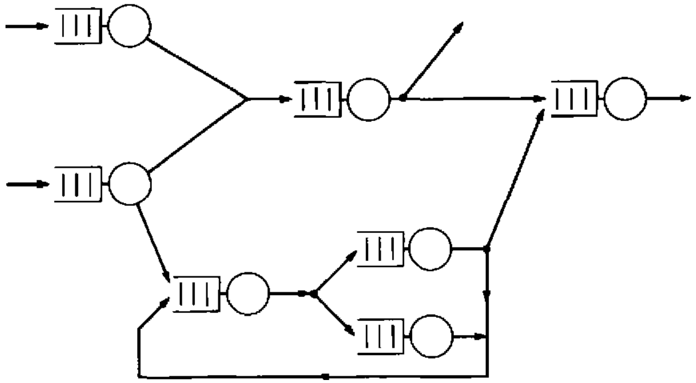
- Use *metamodeling*: Create a statistical “model of the model” mapping inputs to outputs
  - Fast to execute
  - Approximates simulation output

- Ex: For M/M/1 queue (arrival rate λ), estimate $\mathrm{E}_\lambda[L_{4.0}]$
  - Run offline simulations at design points
  - Fit quadratic regression function (or MLP)
  - Can immediately estimate $\mathrm{E}_\lambda[L_{4.0}]$ for new values of λ without needing to simulate



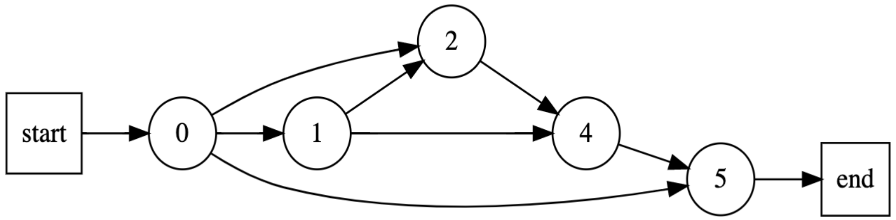
- “Fuzzy” response surface: Gaussian Process (GP) metamodeling

LIMITATIONS OF PRIOR METAMODELING METHODS

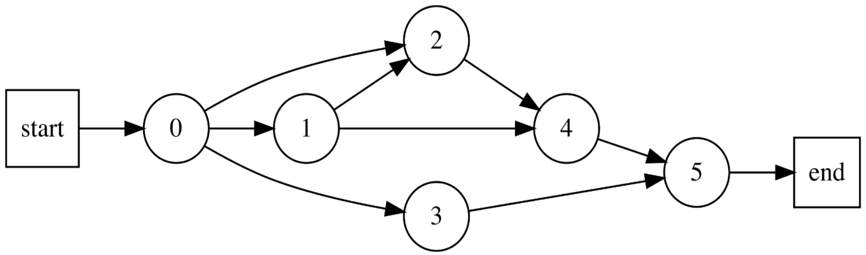


Prior methods ignore simulation structure

- Bottlenecks in queueing networks, critical paths in SANs
- So hard to study impacts of structural changes
  - Example: A traditional metamodel built for SAN1 can't be used for SAN2
  - Can't just feed in adjacency matrix: Permutation-invariance problem [Marti 2019]



$y = f(x_0, x_1, x_2, x_4, x_5)$

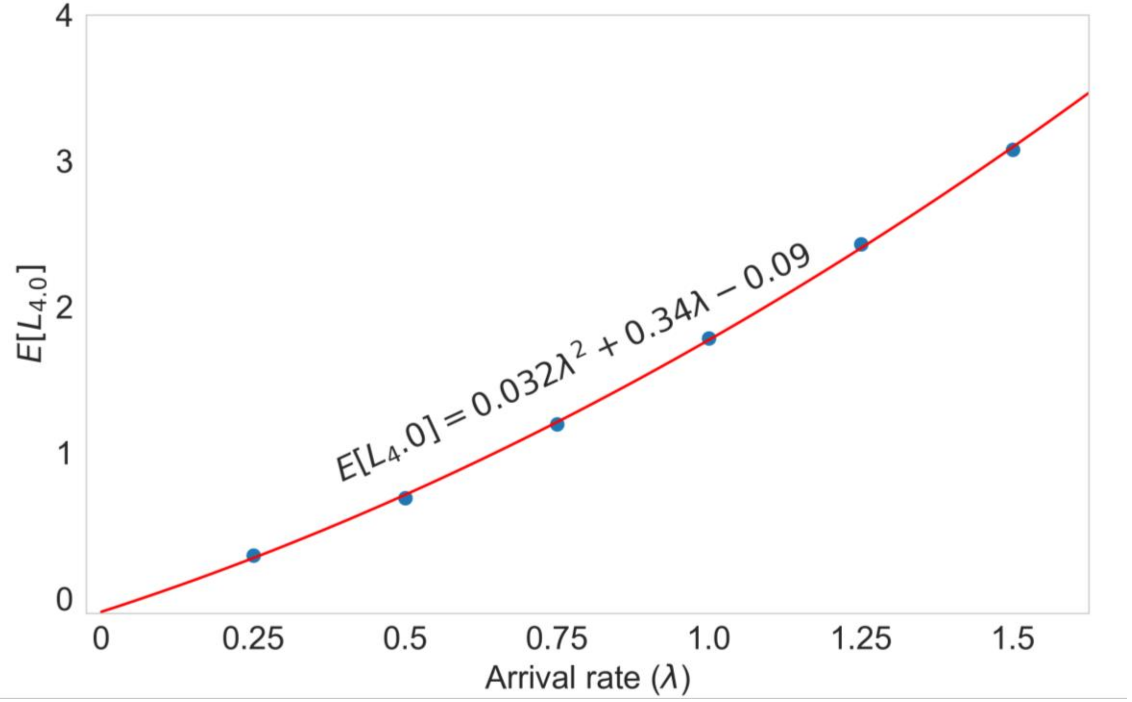


$y = f(x_0, x_1, x_2, x_3, x_4, x_5)$

LIMITATIONS OF PRIOR METAMODELING METHODS

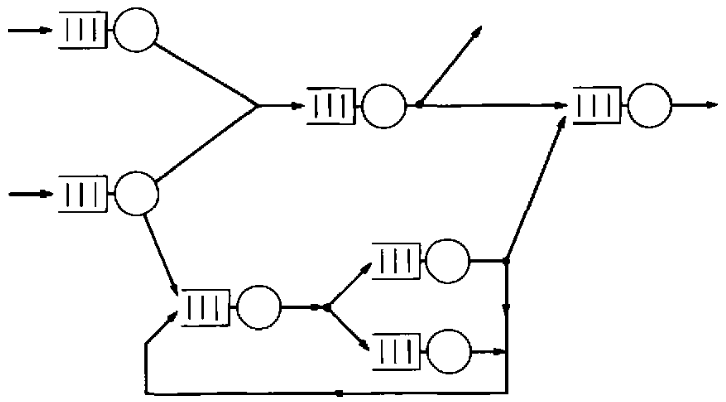
Prior methods only predict real-valued quantities, one per metamodel

- *Original metamodel*: mean of queue-length $L_{4.0}$
- *Now:* 95th percentile of $L_{4.0}$
- *Now:* mean of $L_{8.0}$

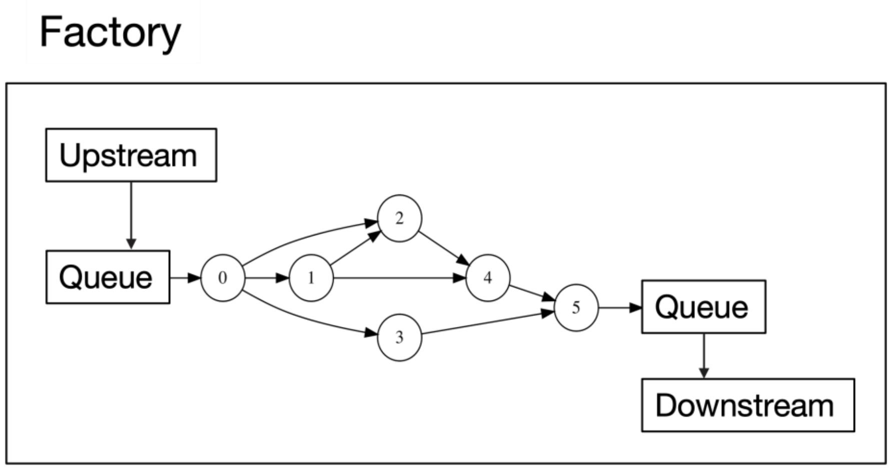


GRAPH NEURAL NETWORKS (WSC 2022)

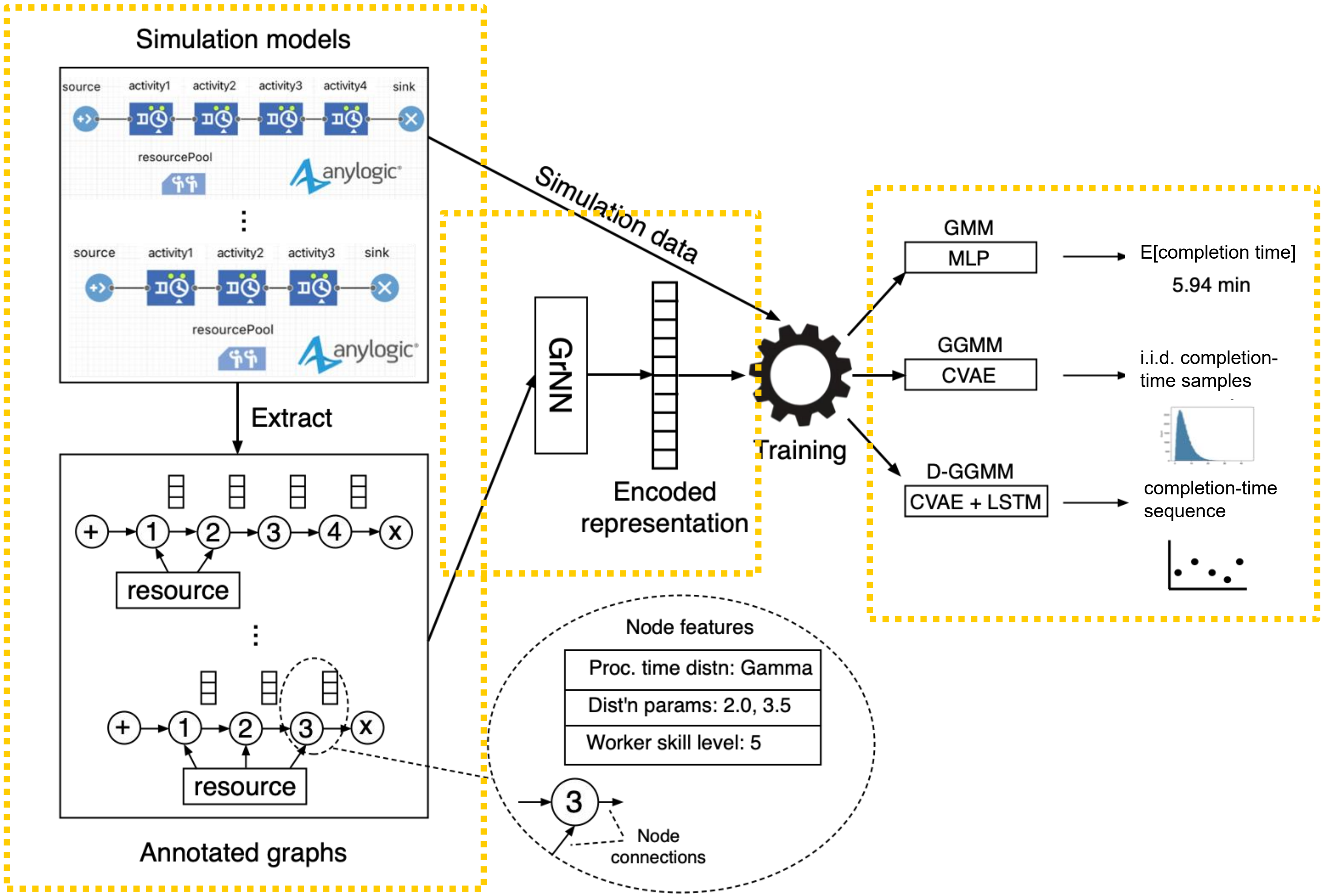
- Treats graph structures as a metamodeling input



- Can easily study the impact of *structural changes*
- Can combine with *generative* neural network components
  - Metamodel can output i.i.d. samples or time series
  - Multiple performance measures from a single metamodel
  - Can provide CIs for point estimates
  - Surrogate model can be embedded in larger model
  - Digital twin applications



GMM OVERVIEW



1. Extract **annotated graphs** from simulations

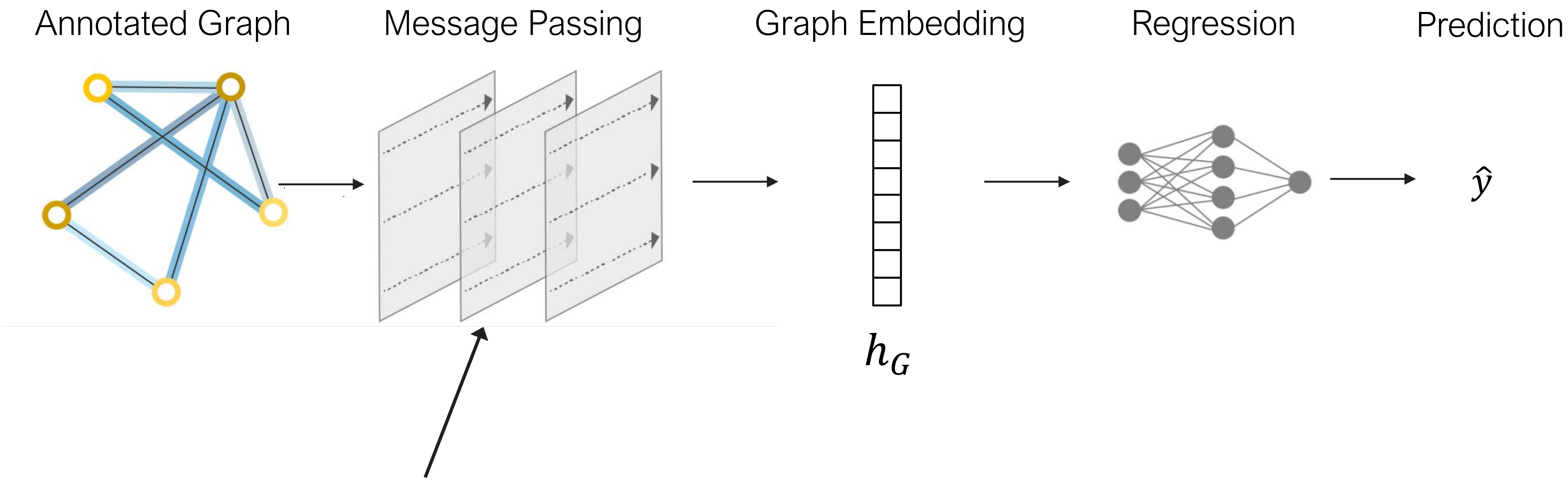
2. **Graph neural net** encodes graph into a “meaningful” embedding

3. **Basic GMM** predicts a numerical performance measure
   - Multi-layer perceptron (MLP)

4. **Generative GMM** generates samples of performance metrics or raw outputs
   - CVAE
   - CVAE + LSTM

Text

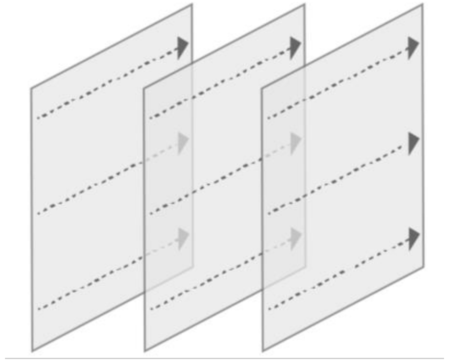
BASIC GMM ARCHITECTURE

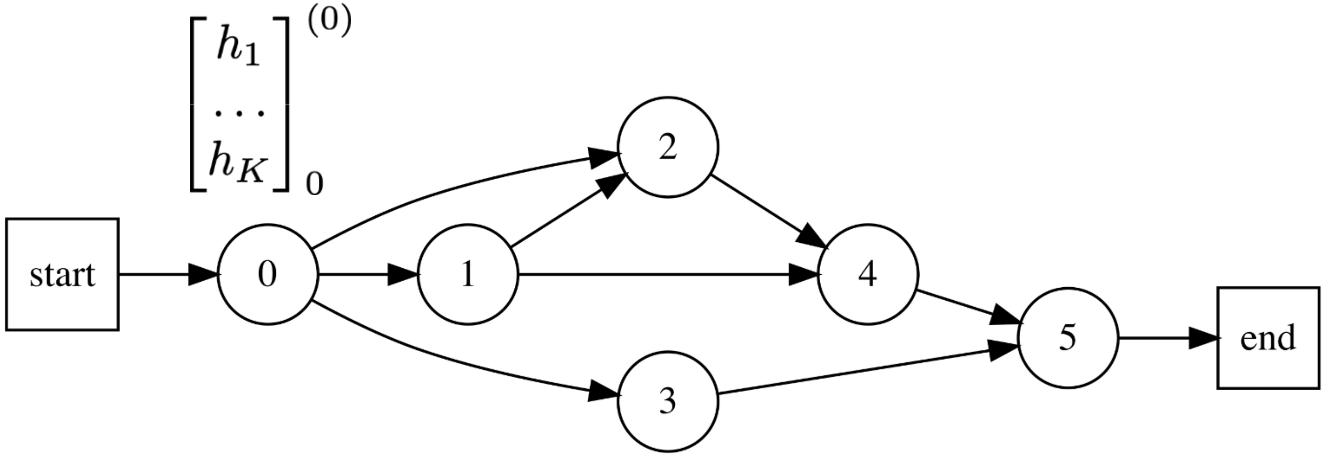


GrNNs use “message-passing” architecture

MESSAGE PASSING

Message Passing

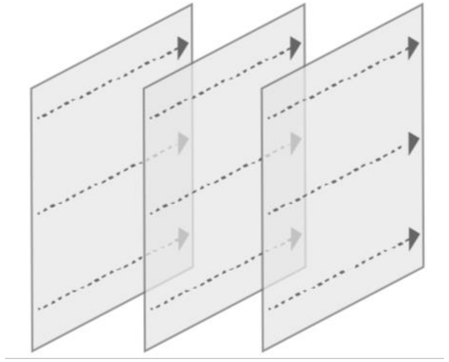


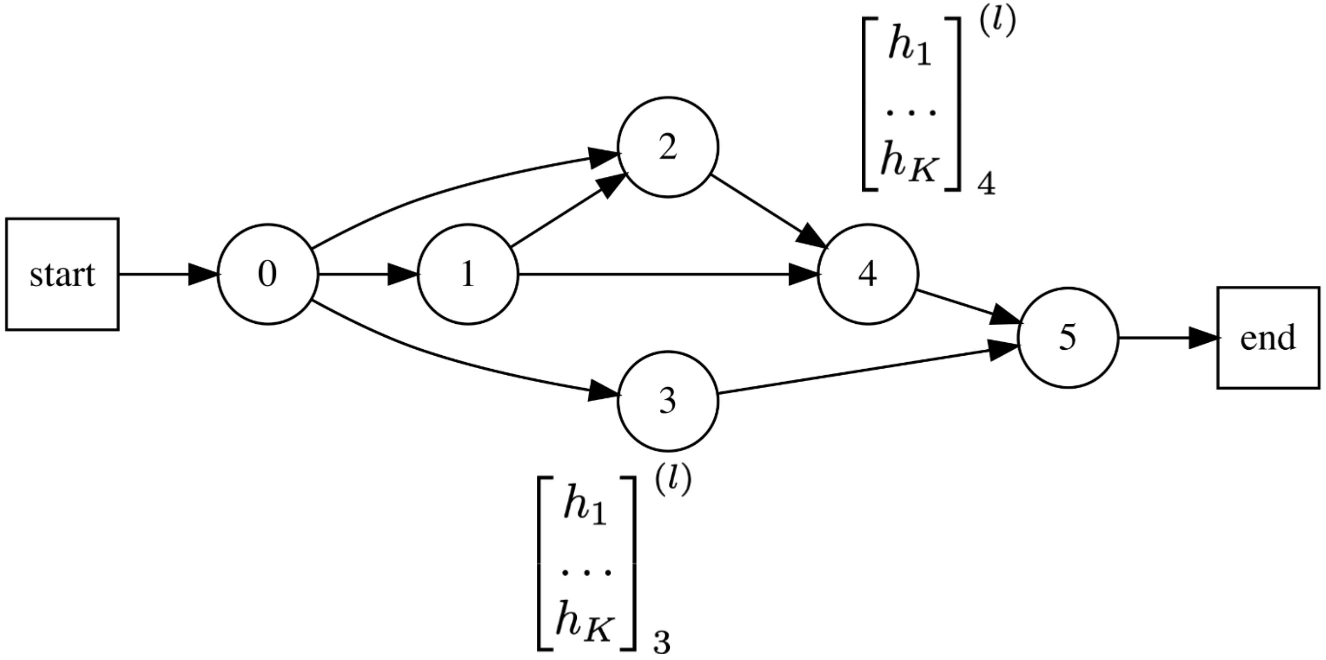


$$h_i^{(0)} = W_1 x_i + b_1$$

MESSAGE PASSING

Message Passing

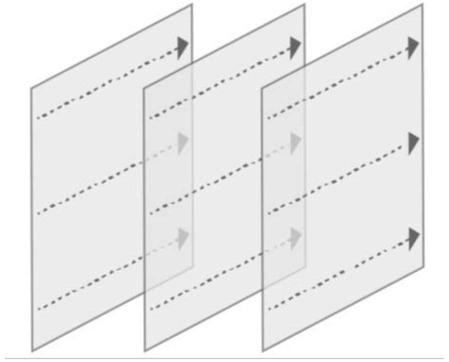


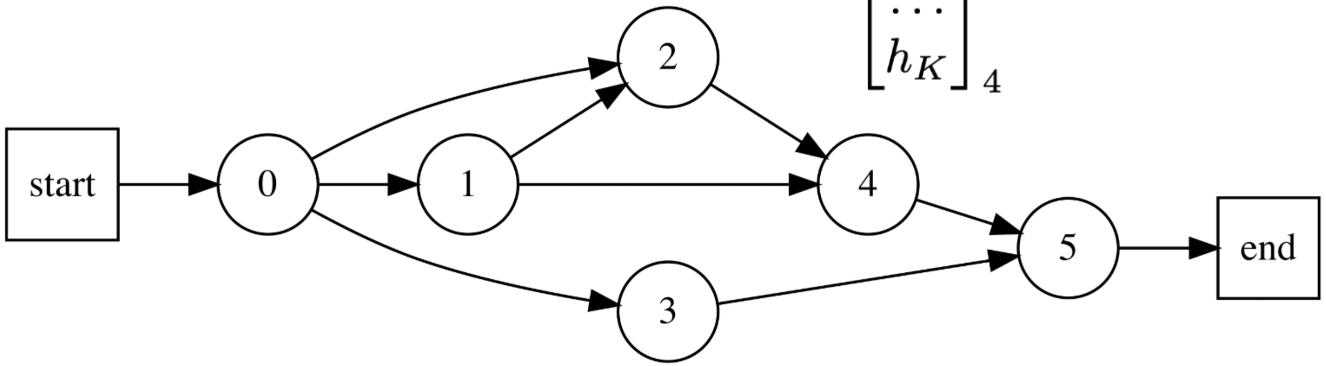


$$h_i^{(l)} = \sigma(W_2 h_i^{(l-1)} + W_3 \sum_{j \in N(i)} h_j^{(l-1)} + b_2)$$

MESSAGE PASSING

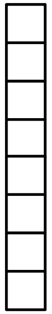
Message Passing





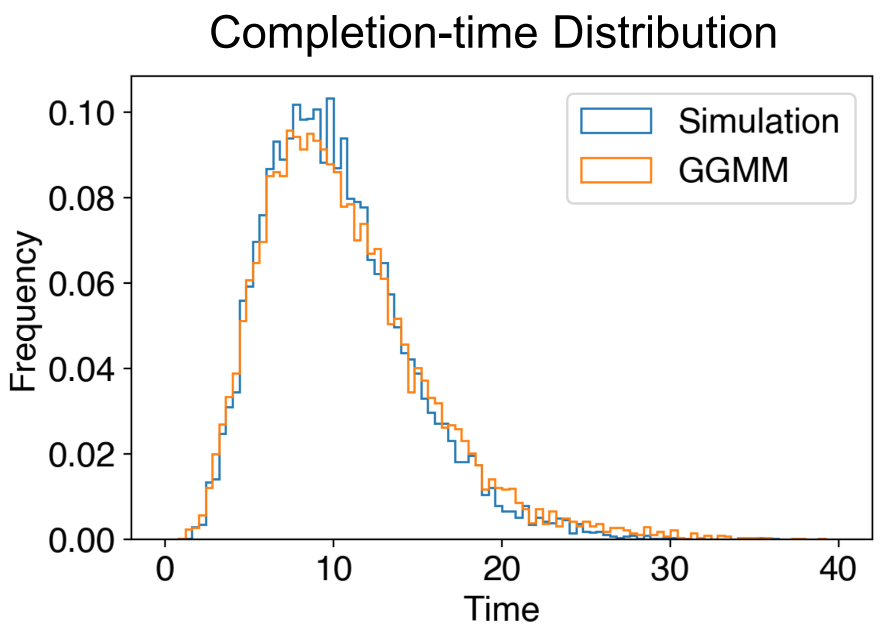
$$\begin{bmatrix} h_1 \\ \dots \\ h_K \end{bmatrix}_4^{(L)}$$

Graph Embedding

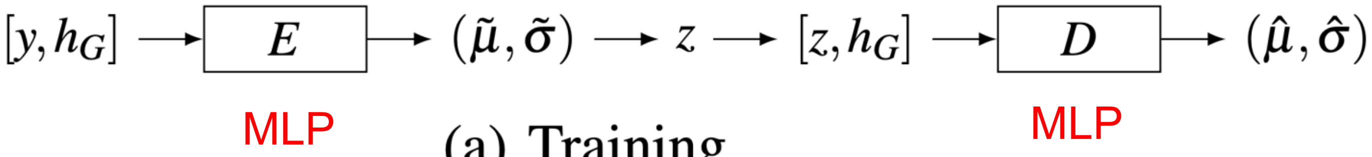


$$h_G = \sum_i h_i^{(L)}$$

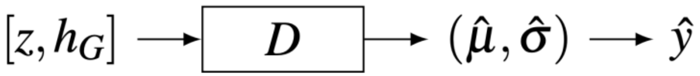
GGMM: COMBINING GMM AND CVAE



- GGMM: Generative GMM
  - GMM + CVAE
  - Use h_G as a condition in CVAE
  - Output = i.i.d. samples of performance measure



(a) Training

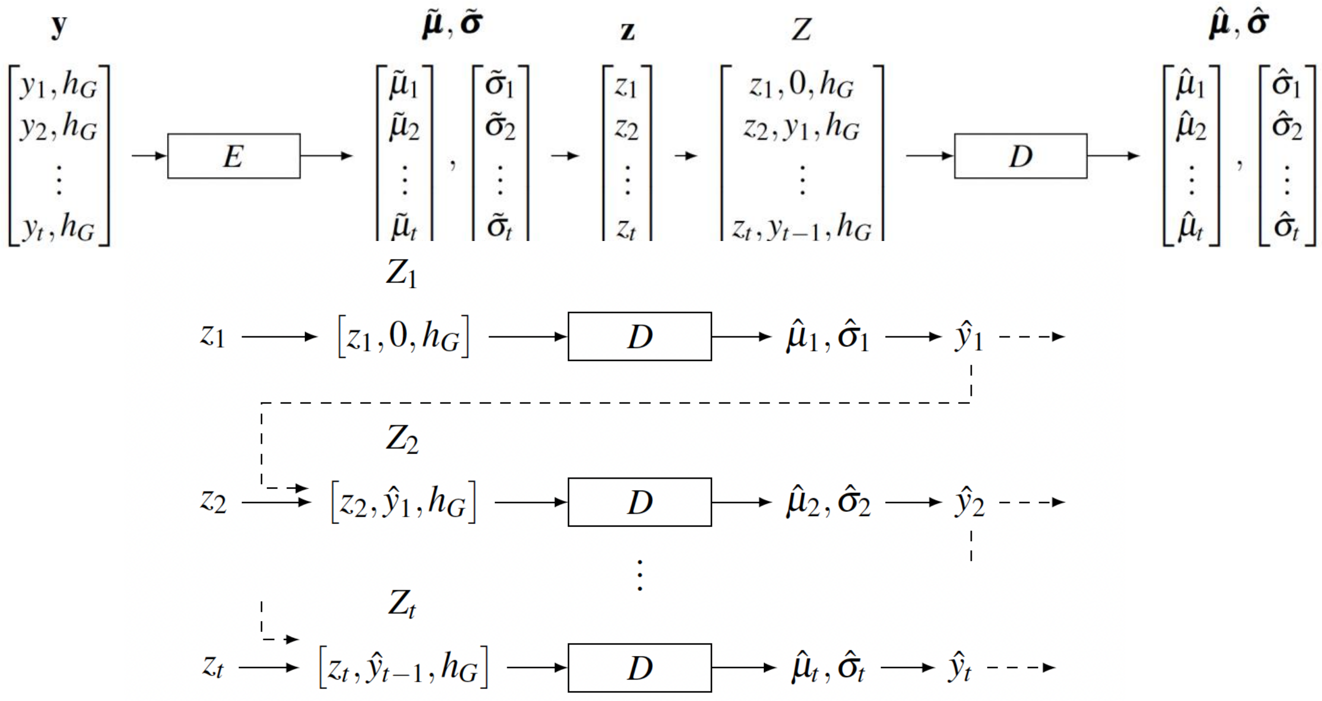


(b) Generation

D-GGMM

- Replace MLP components in CVAE by LSTM components

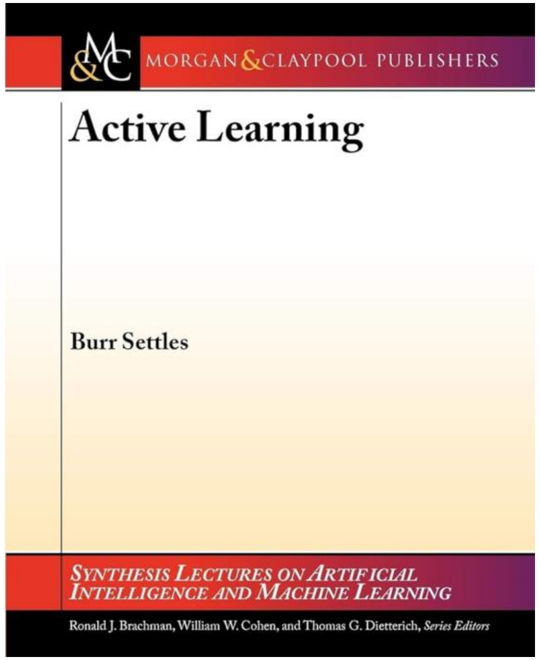
- **D-GGMM**: Outputs stochastic process sample paths



EFFICIENT GMM TRAINING

- **Goal: Reduce # of offline simulation runs**

- **Traditional "active learning" approach in ML**
  - Sequentially choose systems to simulate
  - Choose next system to maximally increase accuracy
  - Uncertainty sampling, version-space methods, etc. for SVM, Random Forest,…



- **Active learning is problematic in neural network setting**
  - Expensive network re-training as each point is added
  - Additional hyperparameters on top of ANN hyperparameters
  - New points might not even be helpful under hyperparameter tuning
    - Ex: 4-layer GMM selects x → train + tune hyperparams → becomes 5-layer GMM → x not useful

- **New HiLo algorithm avoids these deficiencies**
  - Exploits simulation setting
  - Specialized for neural networks

$$[f_\theta(x_i) - f_\theta(x_b)] + \hat{y}_b$$

Computed from difference network trained with CRN

Computed via many simulation replications

OUTLINE

- Background on ML and ANNs
- ANNs for simulation input modeling
- ANNs for simulation metamodeling and optimization
- Other applications of ANNs to simulation
  - Modeling of agent behavior
  - Simulation validation
  - Variance reduction

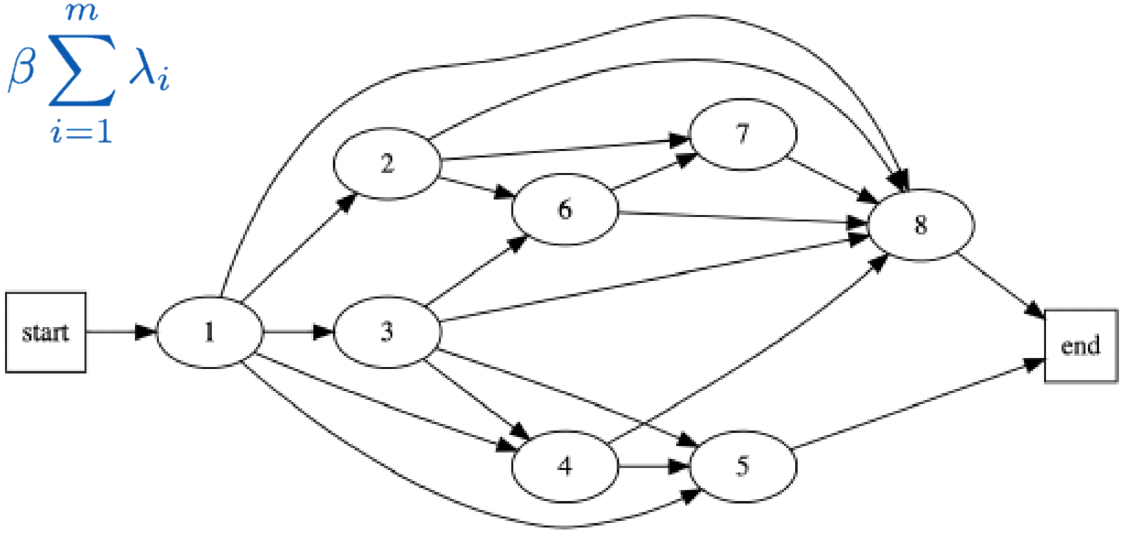
HYBRID OPTIMIZATION WITH GMM'S

- **GMM-based hybrid optimization**
  - GMMs naturally lead to under-explored class of *hybrid* optimization problems
  - Optimize both *graph structure* (discrete) and *model parameters* (continuous)

- **Example: Manufacturing process**
  - Process has *precedence constraints*: child can't start until all parents complete (-> bottlenecks)
  - Incurs costs proportional to process completion time $Y(\lambda, x)$
  - Can pay to speed up work rate or drop edges by buying parts externally

$$\min_{x \in \mathcal{X}, \lambda \in (0,\infty)^n} C(\lambda, x) = E[Y(\lambda, x)] + \alpha \sum_{i=1}^{n} (1 - x_j) + \beta \sum_{i=1}^{m} \lambda_i$$

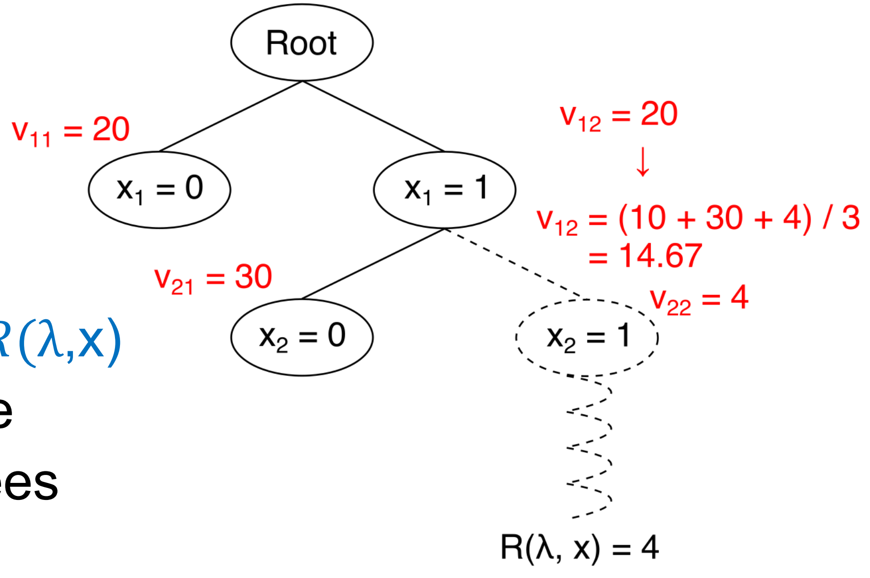
- Challenges:
  - Discrete space is often exponentially large
  - Naïve approach: experts provide promising graph structures (bias, under-exploration)



HYBRID MONTE CARLO TREE SEARCH (WSC 2023)

- **Heuristic but highly scalable**

- **Modified Monte Carlo Tree Search**
  - For efficient exploration of discrete variables
  - Root–to-leaf path = assignment of discrete variables
  - Reward for root-to-leaf path *x* is $R(\lambda^*, \mathrm{x})$ where $\lambda^* = \mathrm{argmax}_\lambda R(\lambda, \mathrm{x})$
  - Reward at leaf guides search towards promising areas in tree
  - Can incorporate R&S "cleanup phase" for statistical guarantees [Boesel et al. 2003]



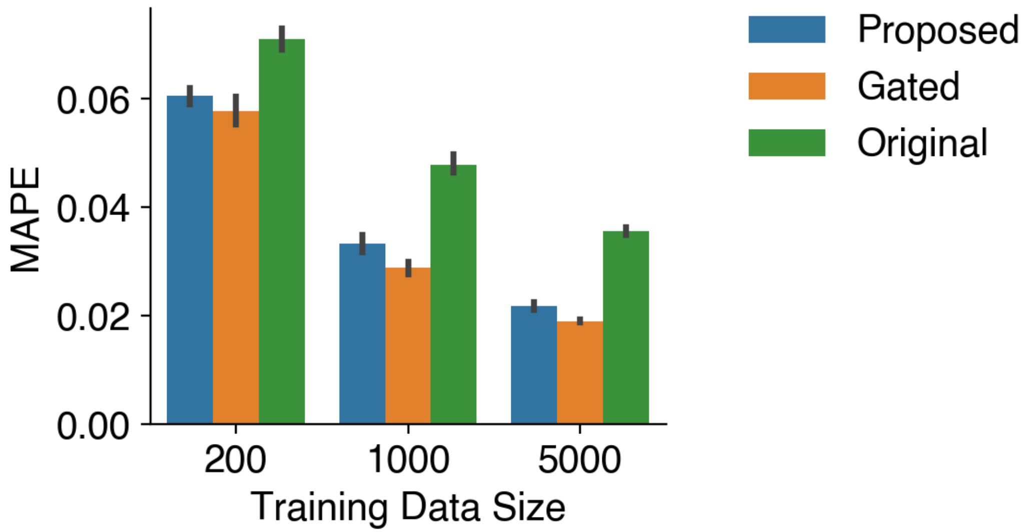
x_i = 0/1 indicator variable for *i*th edge

- **Gradient descent with automatic differentiation for leaf problems**
  - Repurpose built-in AutoDiff libraries used for neural network training

- **Current work:**
  - Exact solution methods based on MILP formulation with specialized solver
  - ANN-guided optimization (like GP-guided optimization but using "neural tangent kernel")

EXACT HYBRID OPTIMIZATION

- **Limitations of H-MCTS**
  - No guarantee of truly optimal solution
- **Exact solution methods**
  - Formulate as a *mixed-integer linear program* (MILP) for *exact* solutions to smaller-scale problems
  - *Revamped GMM architecture* to mimic superior "sequence gated" network but having near-linear form (linear + ReLU)
  - MILP constraints correspond to GMM processing steps
- **Customizing the MILP solver**
  - Structure of MILP leads to slow solution time for off-the-shelf solvers (Gurobi, CPLEX, etc.)
  - Currently developing branch-and-bound method using "affine arithmetic", and parallelization



$$\min_{z^{(\mathrm{n})}, Z^{(\mathrm{e})}, x} \quad c_1 + c_2\hat{y} + g(z^{(\mathrm{n})}, Z^{(\mathrm{e})}, X) \quad \text{s.t.}$$

$$a_i^0 = x_i \text{ if } z_i^{(\mathrm{n})} = 1 \text{ else } 0 \quad \forall i \in [1..N]$$

$$h_i^{(0)} = W_1 a_i^0 + b_1 \quad \forall i \in [1..N]$$

$$a_{j\to i}^{(l),1} = h_j^{(l-1)} \text{ if } Z_{j,i}^{(\mathrm{e})} = 1 \text{ else } 0 \quad \forall l \in [1..L],\ \forall i \in [1..N],\ \forall j \in [1..N]$$

$$a_i^{(l),2} = \left((1-\alpha^{(l)})W_2 \sum_{j=1}^{N} a_{j\to i}^{(l)} + \alpha^{(l)} h_i^{(0)}\right)((1-\beta^{(l)})I + \beta^{(l)}W_3) \quad \forall l \in [1..L],\ \forall i \in [1..N]$$

$$h_i^{(l)} = \max(a_i^{(l),2}, 0) \quad \forall l \in [1..L],\ \forall i \in [1..N]$$

$$h_G = \sum_i h_i^{(L)}$$

$$a^{(3)} = W_4 h_G + b_3$$

$$g_1 = \max(a^{(3)}, 0)$$

$$\hat{y} = W_5 g_1 + b_4$$

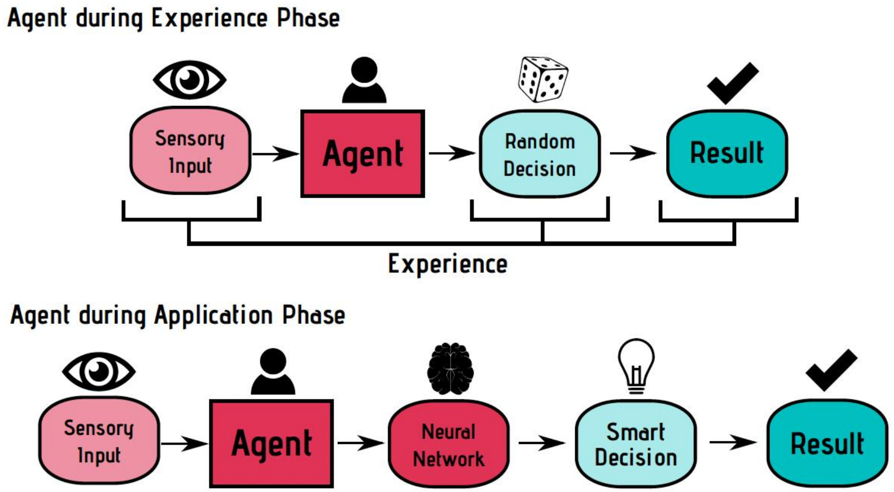
$$(z^{(\mathrm{n})}, Z^{(\mathrm{e})}, X) \in C_j \quad \forall j \in [1..N_c].$$

OUTLINE

- Background on ML and ANNs
- ANNs for simulation input modeling
- ANNs for simulation metamodeling and optimization
- Other applications of ANNs to simulation
  - Modeling of agent behavior
  - Simulation validation
  - Variance reduction

MODELING OF AGENT BEHAVIOR

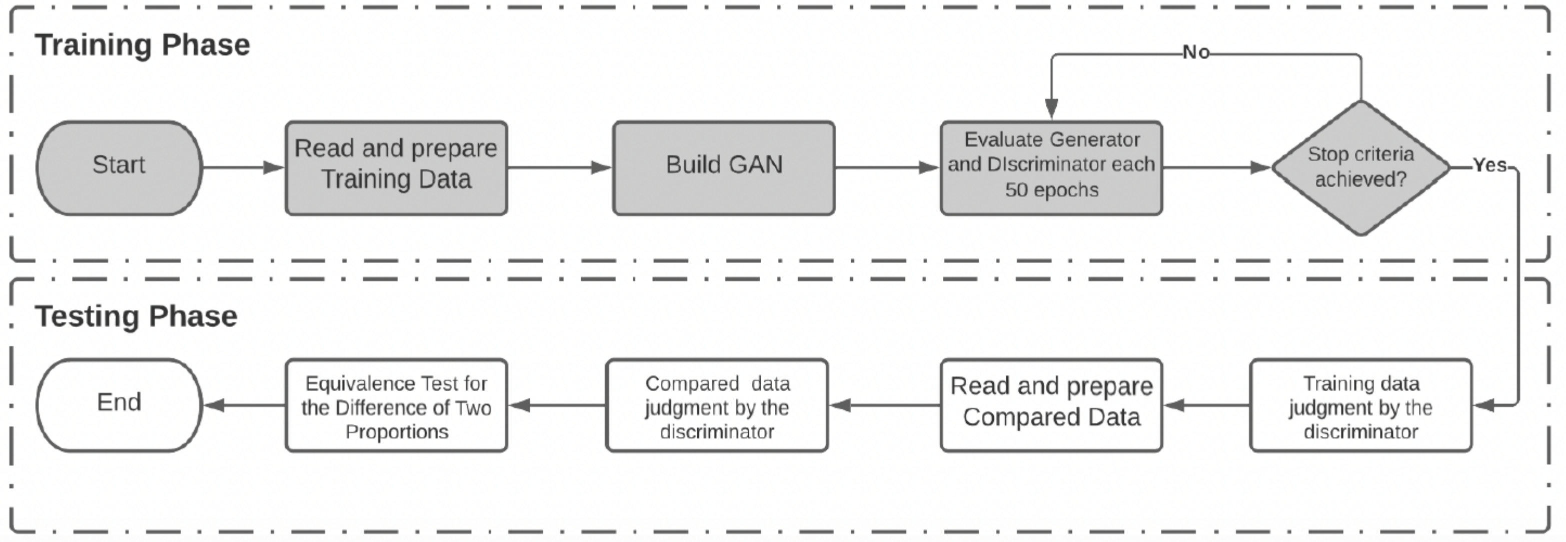
- Replace traditional rule set by MLP [Jaeger ‘19]



- Generative agents [Park+ ‘23]
  - Emergent social behaviours
  - E.g., Valentine’s day party

SIMULATION VALIDATION

- Use GAN to validate simulation [Montevechi+ ‘22]
  - Avoids rigid assumptions of usual statistical tests (normality, simple test statistics, etc.) and can easily handle multiple validation features
  - Train GAN on real-world data
  - Feed real-world data into trained Discriminator and compute rate p_R of correct classifications
  - Feed simulation data into Discriminator and compute rate p_S of correct classifications
  - Test if $p_R - p_S$ is within user-specified tolerance (hypothesis test on diff. of proportions)



VARIANCE REDUCTION

- Idea: Use ANN as a control variate [Lam+ '24]

- Goal: Estimate $\mathrm{E}[f(\theta, Y)]$ where Y is generated by simulation

- Prediction-enhanced Monte Carlo

$$\frac{1}{n}\sum_{i=1}^{n}\left(f(\theta, Y_i) - g(\theta, X_i)\right) + \frac{1}{N}\sum_{j=1}^{N} g\left(\theta, \tilde{X}_j\right) = \sum_{i=1}^{n}\left(f(\theta, Y_i) - C_i\right)$$

  - g is a pre-trained ANN
  - Pairs (X_i, Y_i) are coupled: $X = \phi(Y)$ where X is a vector of features from sample path for Y
  - The i.i.d. random variables $\tilde{X}_1, \dots, \tilde{X}_N$ are independent of (X_i, Y_i)

- HiLo metamodel training can also be viewed as a control-variate-like approach

MANY NEW OPPORTUNITIES FOR RESEARCH

- Use of explainable AI (XAI) techniques to provide insight
  - E.g., SHAP feature-importance metric [Serré+ ‘22]
- Uncertainty quantification
  - E.g., conformal prediction
- Use of LLMs to generate simulation code
- …

MACHINE LEARNING & SIMULATION: FRIENDS!

Machine Learning:

- Statistical model to produce predictions
- Leverages large amount of available data

Adobe Firefly

Simulation:

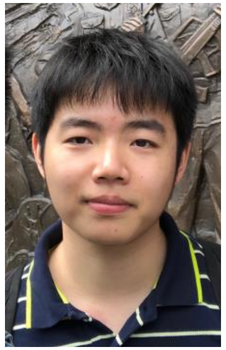
- Mechanistic model of system logic to produce predictions
- Incorporates deep domain expertise (logistics, engineering, healthcare, telecom, computer design, …)

Opportunities to achieve the best of both worlds!

ANNs for

input modeling, metamodeling, simOpt, agent modeling, validation, variance reduction

ACKNOWLEDGEMENTS



- Cen Wang: Co-author on all of my own research
- Emily Herbert: Early contributions to NIM
- Justin Domke and Philippe Giabbanelli: Helpful discussions
- U.S. Army DEVCOM Analysis Center: Support under Contract #W911QX-23-D-009/W911QX-23-F-0115

Backup Slides

Winter Simulation Conference

December 17, 2024

VAE TRAINING

- We train VAE by choosing θ to minimize loss function (via SGD)
- First term: KL-divergence between $Q(z\,|x) = N(\tilde{\mu},\tilde{\sigma}^2)$ and $\mathrm{P(z)} = N(0,1)$
  – 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 $E_z[-\log P(x\,|z)]$ where $z \sim N(\hat{\mu},\hat{\sigma}^2)$
  – The values we sample from $P(x\,|z)$ should look like training data

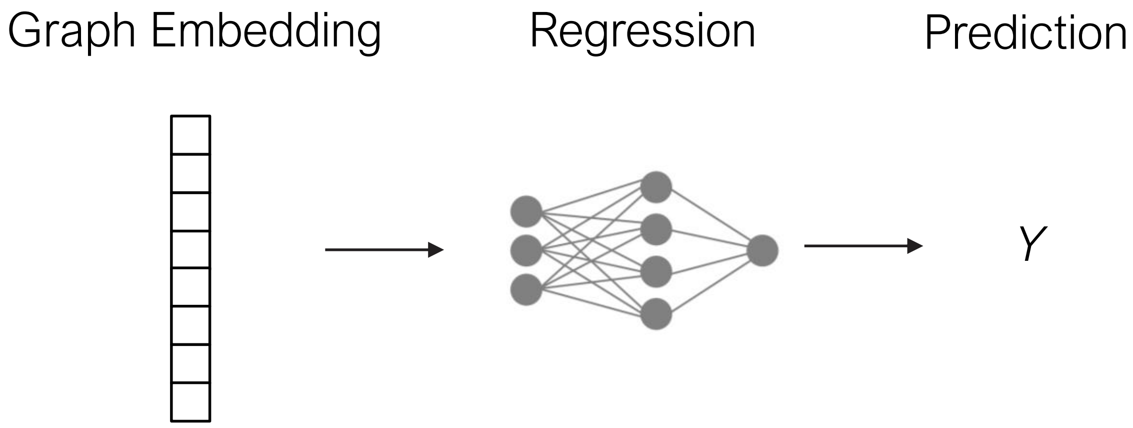
- We train VAE by choosing θ to minimize loss function (via SGD)

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

- First term: **KL-divergence** between $Q(z\,|x) = N(\tilde{\mu},\tilde{\sigma}^2)$ and $\mathrm{P(z)} = N(0,1)$
  – 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** $E_z[-\log P(x\,|z)]$ where $z \sim N(\hat{\mu},\hat{\sigma}^2)$
  – The values we sample from $P(x\,|z)$ should look like training data

REGRESSION



$$\hat{y} = f_\theta(h_G)$$

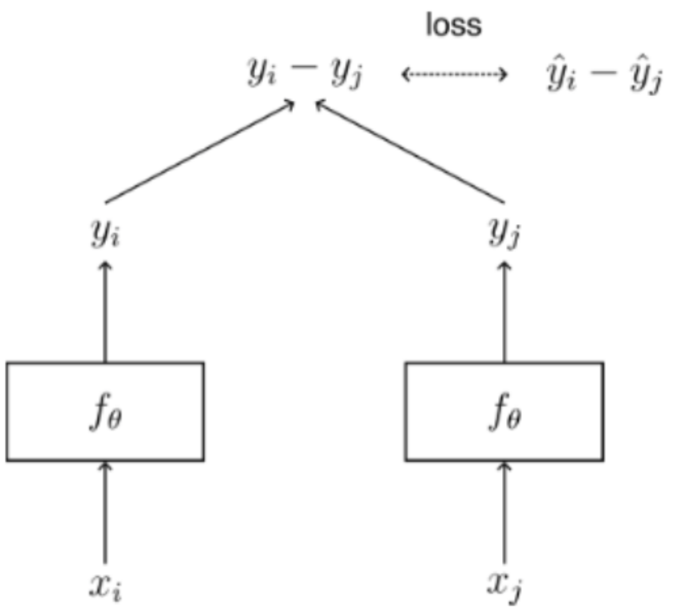
$$L = (\hat{y} - y)^2$$

The weights *W*'s and θ are trained with standard gradient descent (ADAM)

Challenge: "Oversquashing"

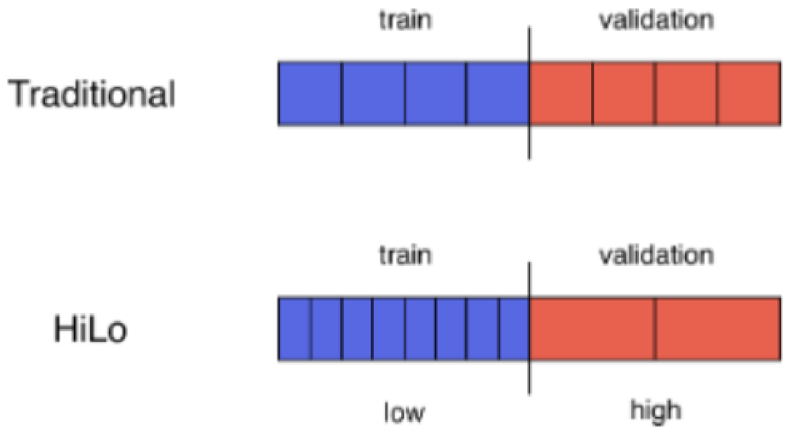
HILO OVERVIEW

- **Modify GMM to predict differences in performance measures**



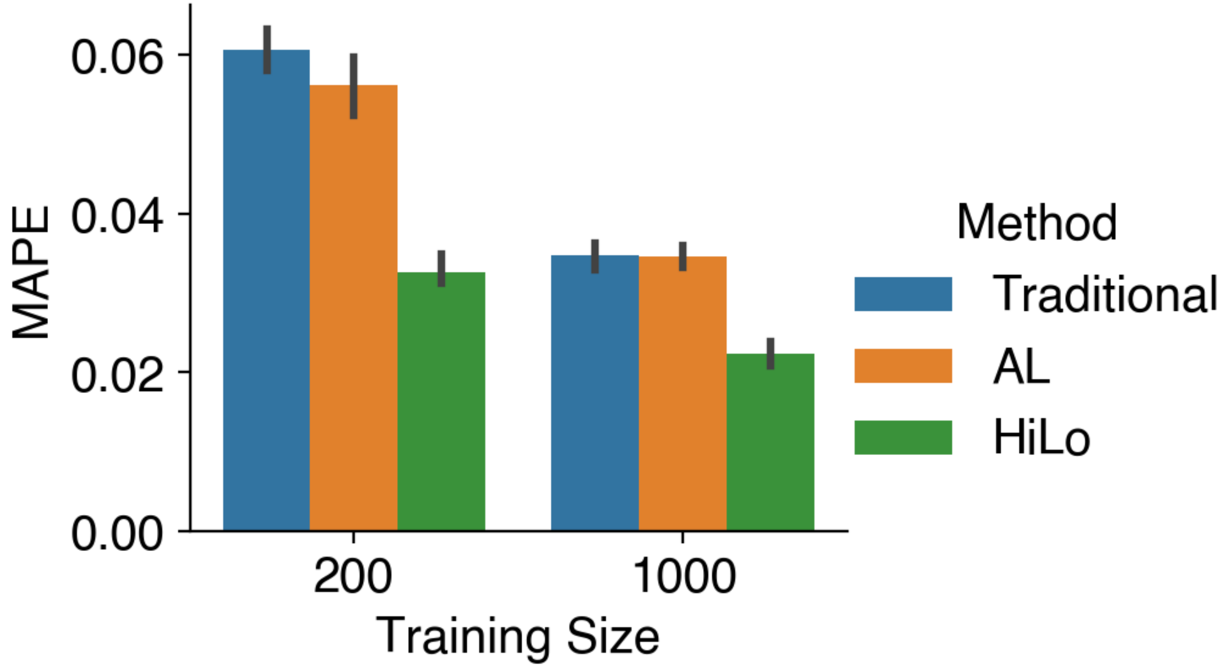
- **Reallocate training and validation replications**
  - High-precision simulation of a few *benchmark* (validation) systems: $\hat{y}_b$ [leverage for prediction!]
  - Low-precision simulation + common random numbers to estimate differences for training systems
  - Final estimate = $[f_\theta(x_i) - f_\theta(x_b)] + \hat{y}_b$

Box width = # of replications for system



HILO, CONTINUED

- **Preliminary empirical study**
  - Initial results: *More effective* than generic active learning methods for ML models

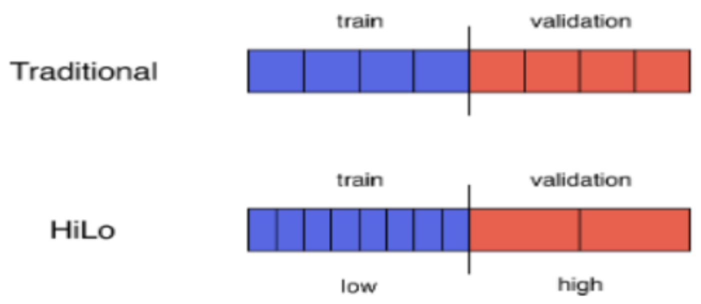


HILO, CONTINUED

- **Preliminary theoretical analysis**
  - Uses theory of *infinite-width neural networks with Gaussian weight initialization* [Jacot+ '18, YangL '21]
  - Limiting GMM is a Gaussian process with *neural tangent kernel* (NTK)

$$K(x, x') = \lim_{|\theta| \to \infty} \nabla_\theta f_\theta(x) \cdot \nabla_\theta f_\theta(x') \text{ a.s.}$$

  - Will help explain *superior properties* of HiLo compared to direct GMM metamodeling and GP metamodeling
- **Ongoing work:**
  - Extend to GMMs with generative components
  - Tune training/validation split



HYBRID MCTS

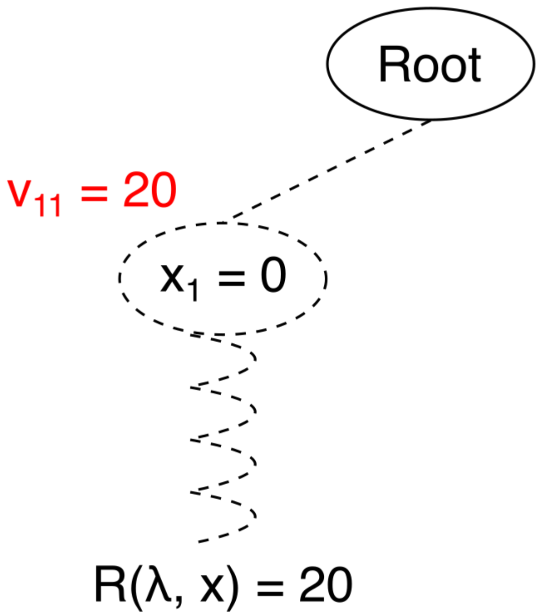
- Traditional MCTS [Fu 2018]: commonly used in AI (AlphaGo)
  - Builds search tree over possible discrete variables (actions)
  - Real number at a terminal leaf is reward for choosing given path

- We replace real number by solution to a continuous optimization problem

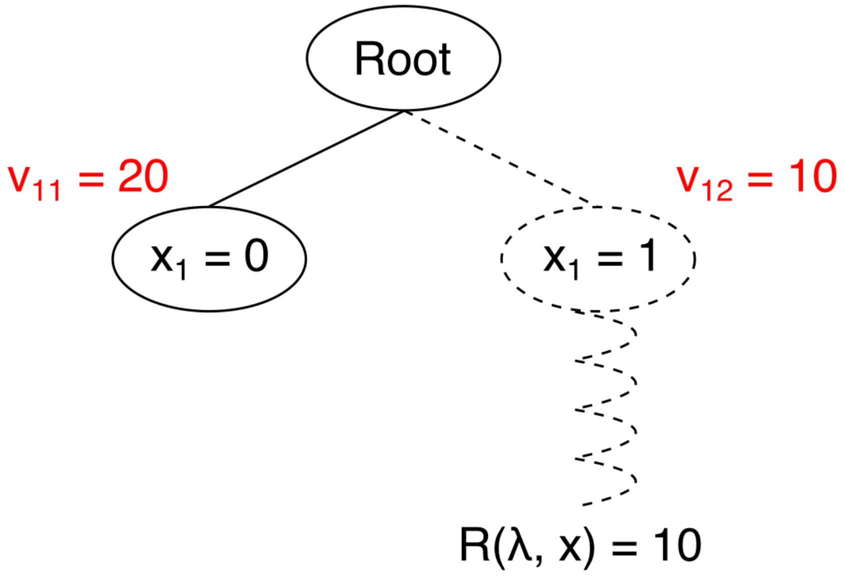
- Four steps for H-MCTS:
  - Selection: probabilistically select a leaf node not fully expanded (via “Gumbel max trick” [Danihelka 2022])
  - Expansion: add a valid child node to the leaf
  - Optimization: randomly set the remaining discrete variables, use gradient descent to optimize continuous variables at terminal
  - Backpropagation: propagate the optimization result to the root, updating selection probabilities in Step 1 (encourage exploration of promising regions)

- Stop when time limit reached

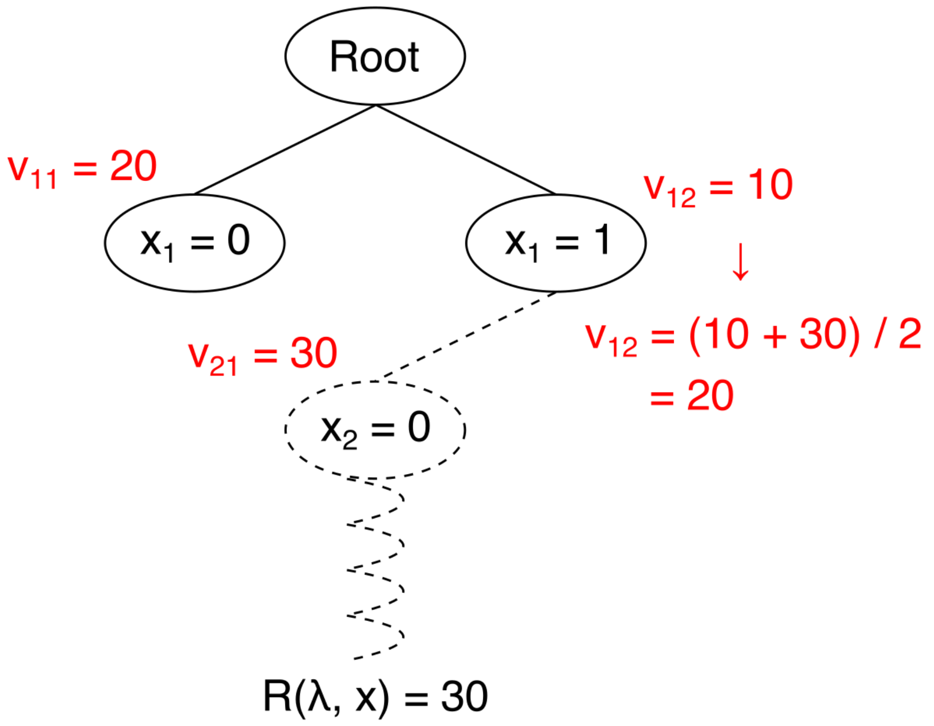
HYBRID MCTS EXAMPLE



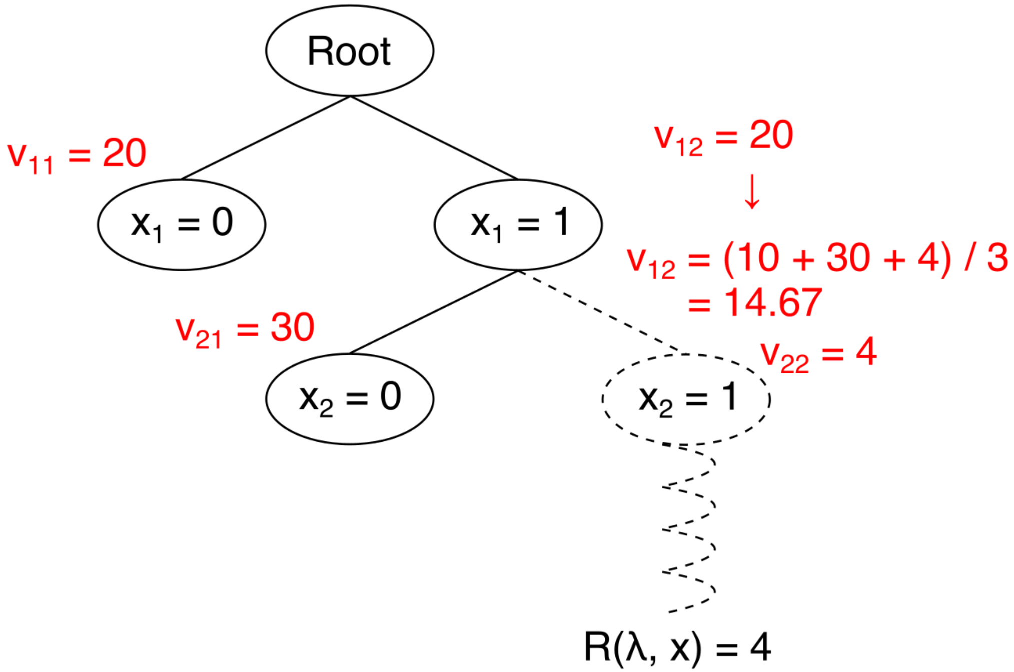
HYBRID MCTS EXAMPLE



HYBRID MCTS EXAMPLE



HYBRID MCTS EXAMPLE



OFFLINE GP-GUIDED HYBRID OPTIMIZATION

- **Prior algorithms are "online" optimization**
  - Build metamodel offline
  - Use it to make online predictions as new simulation models arrive

- **Versus offline optimization: Classic one-shot system design**

- **Idea: Use a Gaussian process (GP) metamodel to guide search for solution**
  - Well-studied for non-hybrid problems (e.g., Hong and Zhang 2021 TutORial)
  - When deciding on next system to simulate, use *UCB criterion* to trade off exploration and exploitation
    - Need GP kernel $K(x, x)$ to compute UCB
    - Traditionally, use, e.g., radial basis function (RBF) kernel on continuous parameters
    - We propose use of *neural tangent kernel*, which can handle (hybrid) annotated graphs