SAND2022-9751C

Learning to Parameterize a Stochastic Process Using Neuromorphic Data Generation

William Severa
Sandia National Laboratories
Albuquerque, NM, USA
wmsever@sandia.gov

J. Darby Smith
Sandia National Laboratories
Albuquerque, NM, USA
jsmit16@sandia.gov

James B. Aimone
Sandia National Laboratories
Albuquerque, NM, USA
jbaimon@sandia.gov

Richard Lehoucq
Sandia National Laboratories
Albuquerque, NM, USA
rblehou@sandia.gov

ABSTRACT

Deep learning is consistently becoming more integrated into scientific computing workflows. These high-performance methods allow for data-driven discoveries enabling, among other tasks, classification, feature extraction, and regression. In this paper, we present a unique approach to solving an inverse problem—determining the initial parameters of a system from observed data—using not only deep learning-powered AI but also simulation data generated using neuromorphic, brain-inspired hardware. We find this approach to be both scalable and energy efficient, capable of leveraging future advancements both in AI algorithms and neuromorphic hardware.

Many high performing deep learning approaches require large amounts of training data. And, while great progress is being made in new techniques, current methods suggest that data-heavy approaches are still best-suited for maintaining critical generalization required for an inverse problem. However, that data comes at a cost, often in the form of expensive high-fidelity numerical simulations. Instead, we make use of recent advances in spiking neural networks and neural-inspired computing wherein we can use Intel's Loihi to compute hundreds of thousands of random walk trajectories. Statistics from these random walkers effectively simulate certain classes of physical processes. Moreover, the use of neuromorphic architectures allows these trajectories to be generated quickly and at drastically lower energy cost. This generated data can then be fed into a deep learning regression network, modified to incorporate certain known physical properties. We find the resulting networks can then determine the initial parameters and their uncertainties, and we explore various factors that impact their performance.

ACM Reference Format:
William Severa, J. Darby Smith, James B. Aimone, and Richard Lehoucq. 2022. Learning to Parameterize a Stochastic Process Using Neuromorphic Data Generation . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Given a collection of observed data, what parameters produced the data? This is the fundamental question of the inverse problem. Some inverse problems seek initial conditions or model forms over parameters, but the core idea of seeking a cause given an observed quantity remains. Inverse problems are notoriously ill-posed. A distribution of an observed quantity (or quantities) is often dwarfed by the number of parameters sought, models often admit unidentifiable pairs, and parameters can sometimes become multimodal where more than one value could equivalently produce observed data.

Methods for approaching inverse problems are wide and varied, changing with the field of study and the ultimate information desired. Turning away from these bespoke methods, machine learning may provide a way to solve particular families of inverse problems. Recently, deep learning has had success in solving electromagnetic scattering problems with numerous direct and physics-assisted learning approaches [1]. Similarly, neural networks have had success solving inverse problems in topological photonics, adhering to physical constraints by coupling with a learned network representing the forward problem [2].

Machine learning has often been used for image classification, however it has also seen success for inverse problems in imaging. For imaging, the inverse problem is the reconstruction of an image given some set of perturbed or noisy measurements. Such inverse imaging problems, with high-profile applications in compression and medical imaging, are well-served by convolutional neural network (CNN) approaches [3].

As a separate domain, the human brain has inspired computational approaches for decades [4, 5]. The concept of using thousands or millions of tightly connected, simple processing units is not new. However, more recently we've seen the introduction of highly performant large-scale neuromorphic systems. These neuromorphic systems now range from specific instantiations of traditional processors [6] to highly exotic devices, such as optical/superconducting approaches [7]. Of particular interest is a class of spiking (or event-driven) large-scale, low-power neuromorphic processors designed around bespoke application-specific integrated circuits (ASICs). This class of hardware platforms includes IBM TrueNorth [8] and Intel Loihi [9].

Spiking neuromorphic computer hardware is beginning to deliver on the promise of effective and low-power brain-inspired computing. Much of their use so far has been relegated to machine learning or artificial intelligence tasks [10–14]. However, there's now also growing interest in their use for other general tasks and, in fact, there have been a number of numerical or scientific workloads that seemingly map well to these esoteric platforms [15–20]. Such generality has motivated investigations into their use alongside traditional processors and accelerators in future high-performance computing platforms [21].

In this work, we look to bridge recent advances in these two separate domains—We examine whether or not a machine learning technique can be used to solve an inverse problem when trained on data generated on a neuromorphic platform. In doing so, we seek to address several aspects:

(1) Deep neural networks and convolutional neural networks are heavily reliant on large amounts of training data. In many domains, synthetic or simulated data is becoming a common proxy for real, collected data. However, the generation of this data can be computationally expensive and energy intensive. By moving data generation to high-efficiency neuromorphic processing, we can hope to decrease the overall burden of synthetic/simulated data generation.
(2) As neuromorphic systems are becoming more closely evaluated for numerical and scientific workloads, there is an obvious need to evaluate the quality of the results produced. Solving an inverse problem helps characterize the ability for the neuromorphic simulation to capture the parameterization of the underlying system.
(3) This work also serves as a first-step, proof-of-concept towards a fully neuromorphic implementation wherein both the simulation and the learning component are instantiated on-hardware. Such a system is beyond the scope of this report, but would be an interesting extension with implications on the feasibility of model-based neuromorphic learning systems.

In this work, we focus on a particular inverse problem: the parameterization of the Ornstein-Uhlenbeck (OU) stochastic differential equation (SDE) given data trajectories. The overall workflow is shown in Figure 1. This equation, sometimes called an autoregressive process, describes a diffusive process that is drawn toward a mean position by a linear-elastic force. This particular SDE is ubiquitous in many fields and has been used, to only name a few, to model neuronal activity [22], pinned molecular motor cargo motion [23], components of epidemics [24], and also in some financial models [25]. Given the success of CNNs in inverse imaging problems and the structure of the input data, we elected to modify and evaluate three common CNN architectures: VGG16 [26], ResNet50 [27], DenseNet121 [28].

While this work focuses on a single inverse problem, parameterizing the OU SDE from data, it does represent a significant first step on the path to a larger goal. Namely, many inverse problems can be solved through particle methods and random walk methods [29–33]. Setting aside machine learning components, such approaches allow us to remove ourselves from the differential equations and allow us to solve for quantities of interest using samples. Sample-focused methods enable inverse problem solving even for highly non-linear problems or those where there are no tractable mathematical and analytic methods. Samples, however, can be expensive to take. By moving the sampling to a neuromorphic framework, we gain efficiency and scaling benefits. In a grander scheme, this single equation, while simple, represents the hardest type of equation to sample on Loihi. Since Loihi is still new and resource-constrained, the number of random outcomes in each time step is limited. The OU SDE by contrast is Gaussian and has an unlimited number of outcomes in a time interval. Hence, although this work is limited in scope, it does provide valuable insight for neuromorphic inverse problem solutions through particle methods.

The remainder of the paper is structured as follows. In Section 2, we discuss the inverse OU problem, and in Section 3 we highlight our method of data generation. We detail our chosen neural networks and results in Section 4, with conventional data being used as a test set in Section 4.3. Lastly we provide some discussion and closing remarks in Section 5.

2 THE INVERSE ORNSTEIN-UHLENBECK PROBLEM

Neuromorphic computer hardware can efficiently implement diffusion through random walks [34] and those neuromorphic random walks are sufficient for solving a family of steady-state PDEs [35, 36]. Further, in this case these samples and random walks generated appear to adhere statistically to their expected distribution [37]. The random walks simulated on Loihi through this method yield *density* information. A set number M of random walks are started at the same location, and they evolve over time. At any given time step, the spike count per neuron is interpreted as the number of walkers on a particular position.

As previously mentioned, the samples returned can be thought of as as approximations to some real process. Much like in real experimentation, we want to collect our data and parameterize the underlying process. This is the fundamental concept of the inverse problem. There are several approximations made in mapping an SDE or stochastic process to a discrete time Markov chain (DTMC) on Loihi. Apart from discretizing space and time, assumptions have to be made to construct a transition matrix and hardware limitations constrain random number precision (see [36, 37]). Like in many real-world experiments, data generated reflects that of a true process or event, but is masked by several factors. While we will stop short of describing hidden and observable processes and discussing the worthiness of Kalman filters, we do claim an opportunity to not only test whether using machine learning can parameterize stochastic processes, but also whether or not the data generated by the Loihi approximation is good enough to be trusted. If we can successfully parameterize the stochastic equation from the Loihi data, then we can place more trust in the samples generated.

Due to the previously mentioned multi-field relevance of the problem, we confine our study to the Ornstein-Uhlenbeck (OU) equation. Apart from relevance to many disciplines, this equation is also useful for study since it has a known solution and known short- and long-term statistics, making parameterization by hand possible if necessary. The OU equation is a stochastic process written as an SDE and describes the position of a particle over time, $X(t)$. It has three main parameters: k, the spring constant; D, the diffusivity of

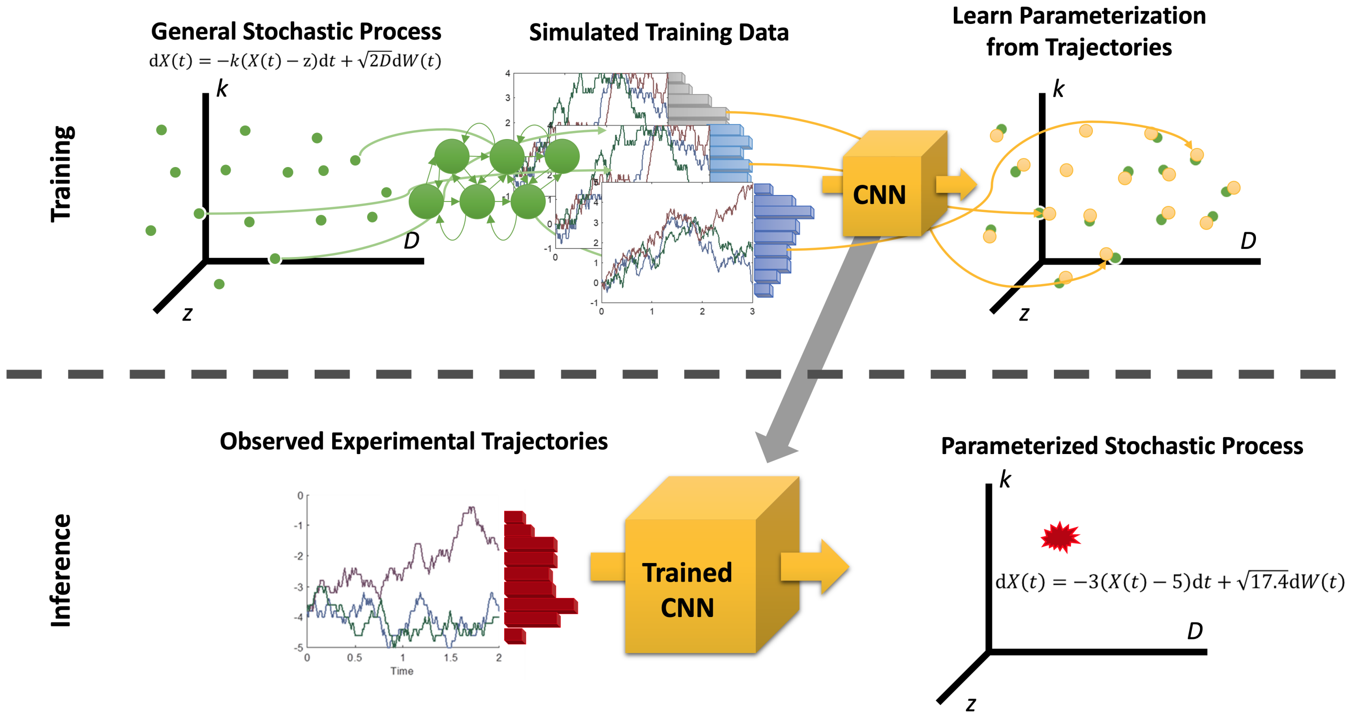

General Stochastic Process
$dX(t) = -k(X(t) - z)dt + \sqrt{2D}dW(t)$
Simulated Training Data
Learn Parameterization from Trajectories
Training
CNN
k
D
z
Observed Experimental Trajectories
Parameterized Stochastic Process
Inference
Trained CNN
Time
$dX(t) = -3(X(t) - 5)dt + \sqrt{17.4}dW(t)$

Figure 1: Overview of the OU Inverse problem setup. The CNN model is trained using random walks in simulated data under a variety of conditions (see Sec. 3); at inference, that model would be used on experimental data instead. For our application, we restrict ourselves to the walker counts and do not require the paths themselves.

the particle; and z, the mean position of the process. Letting $W(t)$ represent a standard white noise process, the one-dimensional OU equation is

$$dX(t) = -k\,(X(t) - z)\,dt + \sqrt{2D}dW(t). \tag{1}$$

This notation for SDEs is shorthand for the implicit integral equation

$$X(t) = X(0) - k\int_0^t (X(u) - z)\,du + \sqrt{2D}W(t).$$

This equation can be solved for $X(t)$, yielding

$$X(t) = X(0)e^{-kt} + z\left(1 - e^{-kt}\right) + \sqrt{2D}\int_0^t e^{-k(t-u)}dW(u),$$

where the final term is a stochastic integral.

By hand, the average initial slope of the sampled paths allows us to infer k, while the mean and variance of the asymptote of the sampled paths allow us to determine both z and D/k. Our version of the inverse problem takes density sampled Loihi data from (1) and employs convolutional neural networks to recover the parameters k, D, and z. We will use our generated data as training and validation data for our three CNNs; we additionally test against data generated conventionally in MATLAB, see 4.3. In the following section, we will briefly discuss the generation of data from (1) on Loihi.

3 DATA GENERATION

The most straightforward way to generate data trajectories from (1) is to employ some standard SDE discretization scheme, where one discretizes time and samples the position value along a time series. Some examples of these schemes are Euler-Maruyama and Stochastic Runge-Kutta. Loihi, however, cannot currently sample a position value from an arbitrary distribution. Instead, we must approximate our process with a discrete time Markov chain (DTMC).

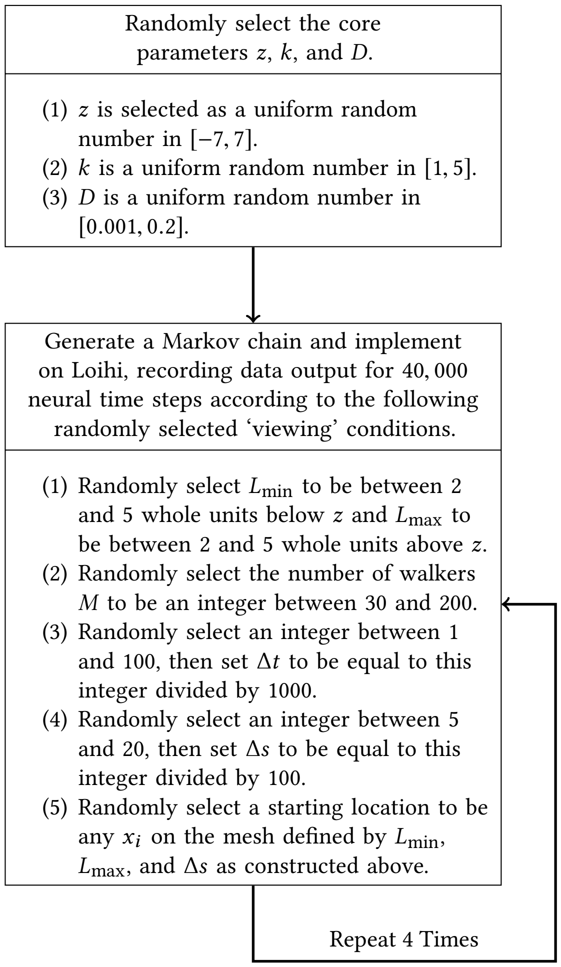
A DTMC can be created from (1) by discretizing the domain of $X(t)$, the real line, into discrete buckets of a fixed size, Δs. Then, as hardware restrictions force us to have a finite space, we must truncate the line by use of a maximum and minimum value, $L_{\min}$ and $L_{\max}$. We treat our state space as the midpoint of all the discrete buckets we have made. From here, we consider the Euler-Maruyama discretization of (1) with time step size Δt:

$$X\,(t + \Delta t) = X(t) - k\,(X(t) - z)\,\Delta t + \sqrt{2D}\zeta, \tag{2}$$

where ζ is a normal random variable with mean zero and variance Δt. This discretization is a probability distribution for $X(t + \Delta t)$ given $X(t)$, and this distribution is independent of t. We can integrate the associated probability density function to obtain the probability of starting at any given state space location and landing anywhere in another bucket in our discretization in the time interval Δt. We use these obtained values as our transition matrix for the DTMC. For more information, see [37].

Note, however, that there are many approximations to the original process in obtaining this transition matrix. Not only are we discretizing the original SDE, but we are also implying that transitions

Figure 2: Process for generating DTMC on Loihi.



Randomly select the core parameters z, k, and D.

(1) z is selected as a uniform random number in $[-7, 7]$.
(2) k is a uniform random number in $[1, 5]$.
(3) D is a uniform random number in $[0.001, 0.2]$.

Generate a Markov chain and implement on Loihi, recording data output for 40,000 neural time steps according to the following randomly selected 'viewing' conditions.

(1) Randomly select $L_{\min}$ to be between 2 and 5 whole units below z and $L_{\max}$ to be between 2 and 5 whole units above z.
(2) Randomly select the number of walkers M to be an integer between 30 and 200.
(3) Randomly select an integer between 1 and 100, then set Δt to be equal to this integer divided by 1000.
(4) Randomly select an integer between 5 and 20, then set Δs to be equal to this integer divided by 100.
(5) Randomly select a starting location to be any x_i on the mesh defined by $L_{\min}$, $L_{\max}$, and Δs as constructed above.

Repeat 4 Times

occur from midpoint to midpoint, even though the probabilities are calculated based on midpoint to intervals.

Once the transition probabilities and the state space of the Markov chain have been determined, we need only initialize a set of M walkers on some starting location $X(0)$ and record their evolution over time. On Loihi, we implement the DTMC as in [36] using the density algorithm originally from [34]. Essentially, a collection of neurons takes a random walker in the form of a spike, and routes it to any one of a number of output neurons by way of a mutually exclusive probability draw.

Our data have parameters generated partially through a grid search and partially through a random search. The grid search was used initially but deemed too inefficient. This portion of the dataset is unbalanced in k, D, and z and is only used for training (i.e. not validation nor testing). We determine our random parameterizations using the process shown in Fig. 2; the process can be repeated as long as desired.

We selected arbitrary ranges for our parameters with the only goal of having reasonable fluctuations. The parameter z is completely arbitrary as it only centers the process. Fluctuations can be controlled on the long-term scale by reducing the ratio of D/k or from a time step-to-time step scale by choosing a small Δt or large Δs.

Even within our pre-defined ranges, certain combinations of parameters and viewing conditions are non-viable. The determination of which combinations are non-viable is complex, and a separate report has included some investigation into this phenomenon [37]. For the results presented in this paper, we have generated 41,251 valid data points on Loihi, where a data point is the evolving density of one (k, z, D) triple under a single viewing condition.

We have previously determined that data generated in this manner can have strong deviations from the expected distribution, but on the whole generate samples that are statistically expected [37]. While this means that samples are not too bad on average, it is not clear whether samples are useful for all cases. If we have success at the inverse problem through our machine learning task, we can increase our trust in Loihi synthetic energy efficient data generation.

4 NEURAL NETWORK SOLUTIONS

Deep learning approaches have been studied extensively for solving inverse physics problems (see [1–3], for example). While there are many interesting challenges and trade-offs in this area, our main objective was to apply popular image classification CNN architectures to stochastic process data in order to evaluate their capability. Our data is structurally similar to images, and we believe that an evaluation of largely unmodified off-the-shelf methods will provide a strong baseline of performance.

4.1 Network Design and Results

We evaluated three common convolutional neural network architectures: VGG16 [26], ResNet50 [27], DenseNet121 [28]. These three networks represent popular algorithmic features likely to appear in a successful approach. However, none of these methods were originally designed for this task, and so we recognize that there are some shortcomings with these choices, see Sec. 5. To help explore the training data requirements, we studied two data conditions, a base configuration with 25,874 samples and an expanded configuration with 37,554 samples. Both conditions used the same 3697 sample validation set. Viewing condition repetitions were (see Fig. 2) not split across training and validation sets.

Convolutional neural networks exhibit sensitivity to a large number of hyperparameters. To help mitigate this, we used a multi-node GPU cluster to perform automated hyperparameter optimization. Hyperparameter selection was based on a tree-structured parzen estimator (TPE) method implemented by the optuna package [38]. Our search seeks to optimize many of the standard hyperparameters such as learning rate, optimizer choice, number of training epochs, etc. An unused search parameter was removed from the search space of the Extended condition. We used mean squared error for the loss function and scheduled 50 evaluations for each network type, limiting to a maximum of 5 concurrent runs each.[1]

Best validation absolute errors are listed in Table 1 and shown in Figure 3. Overall, we see that DenseNet121 performs the best,

[1]For unknown reasons, possibly due to scheduling issues, a small number of runs failed to complete.

	Parameter	DenseNet121	ResNet50	VGG16
Base	D	0.032	0.035	0.035
	k	0.981	1.040	1.083
	z	0.397	0.527	0.518
Expanded	D	0.022	0.022	0.024
	k	0.383	0.496	0.491
	z	0.179	0.185	0.171

Table 1: Minimum mean absolute validation errors for the three network models with base and expanded training sets.

with minimum validation error being 0.022, 0.383, and 0.179 for D, k and z respectively. We note, however, that the difference between network models is smaller with the expanded dataset. As expected, all networks perform considerably better with the additional training data. Interestingly, we notice that z contributes considerably to the total error, despite the fact that a domain expert could estimate z directly using the final time step of a sufficiently long run.

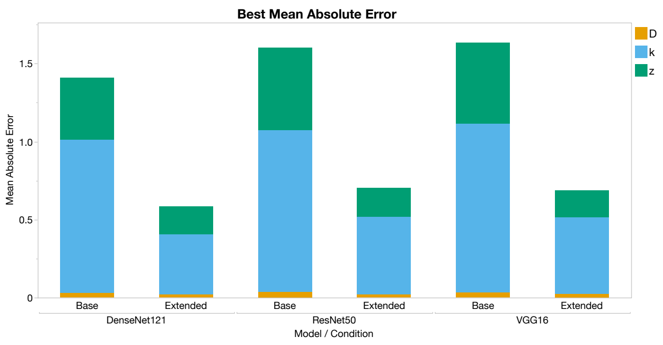


Figure 3: Best validation mean average error for D, k and z after 50 trials for three common neural network architectures. Note that the best error for each parameter may come from more than one network.

4.2 Network Training

Here we examine the 'best' trials for each model/condition pair, where 'best' is measured by total validation loss. The losses during training are shown in Figure 4. In all cases, we see that the training loss is considerably lower than the validation loss, likely due to overfitting and training set imbalance. In particular, we notice that the total training loss for the VGG network is considerably lower than the other networks, despite similar validation losses. Moreover, there appears to be more variation in the validation loss compared to the training loss and this is worsened for the DenseNet and ResNet topologies. This may be a byproduct of the longer training time for the VGG network or this may reflect our omission to adjust momentum hyperparameters. As expected, the difference between training and validation loss is less with the expanded training dataset. This suggests that there may yet be improvements available with even larger datasets.

	Parameter	DenseNet121	ResNet50	VGG16
Base	D	0.08	0.05	0.03
	k	1.09	1.08	1.15
	z	0.50	0.66	0.49
Expanded	D	0.03	0.03	0.03
	k	0.54	1.12	0.72
	z	0.20	0.16	0.14

Table 2: Error on the MATLAB-generated data for the lowest validation error models in Base/Expanded conditions for DenseNet121, ResNet50, and VGG16. Within each condition, each column corresponds to the same single model.

4.3 Network Testing against Conventional Data

Convergence results of the DTMC approximation to the stochastic process are known [39]. Nonetheless, it is of interest to consider how our network would perform when tested against data simulated conventionally through a standard technique like Euler-Maruyama.

We simulated conventional data by using the Euler-Maruyama discretization process (2) as follows. First, we follow the process outlined in Sec. 3, fixing $M = 100$, and generating 4163 samples of the evolving histogram count for (k, z, D) tuples that can be accomplished on Loihi. Crucially, for these Loihi-possible pairs, we generate the trajectories using Eq. 2 exclusively and do not integrate creating a DTMC. Using the same (k, z, D), Δs, Δt, $L_{\min}$, and $L_{\max}$ for each sample we generate trajectories in MATLAB.

Through this process, we generated data using MATLAB matching the 4163 tuples and viewing conditions samples and tested this MATLAB data using the best models measured by total validation loss. The (k, D, z) error for models trained on base/expanded datasets are listed in Table 2.

5 DISCUSSION

The results presented show that it is indeed possible to recover the OU process using off-the-shelf deep learning methods. However, we also recognize that the off-the-shelf methods used are likely limited due to their original design decisions. All three methods were designed for image processing which is an application that is only structurally similar to our application here. In practice, our application faces a challenge that is data-poor and high-dimensional; determination of the underlying process is best determined using the entire process. In contrast, image classification or object localization can often be achieved despite aggressive downsampling. We hypothesize that low-data methods, such as Bayesian neural networks, combined with very large receptive fields could produce higher-performing algorithms. This may be accomplished through techniques such as atrous convolutions. We further recognize that such an approach may benefit generalization as well.

The overall poorer performance of the network when tested on conventional data could be due to overfitting as we hypothesized in Sec. 4, and we expect the performance to drop off significantly as the data moves further away from the training domain. Another possibility is that the DTMC data generated on neuromorphic hardware introduces additional error from low precision calculations. The study of whether or not neuromorphic generated DTMC data

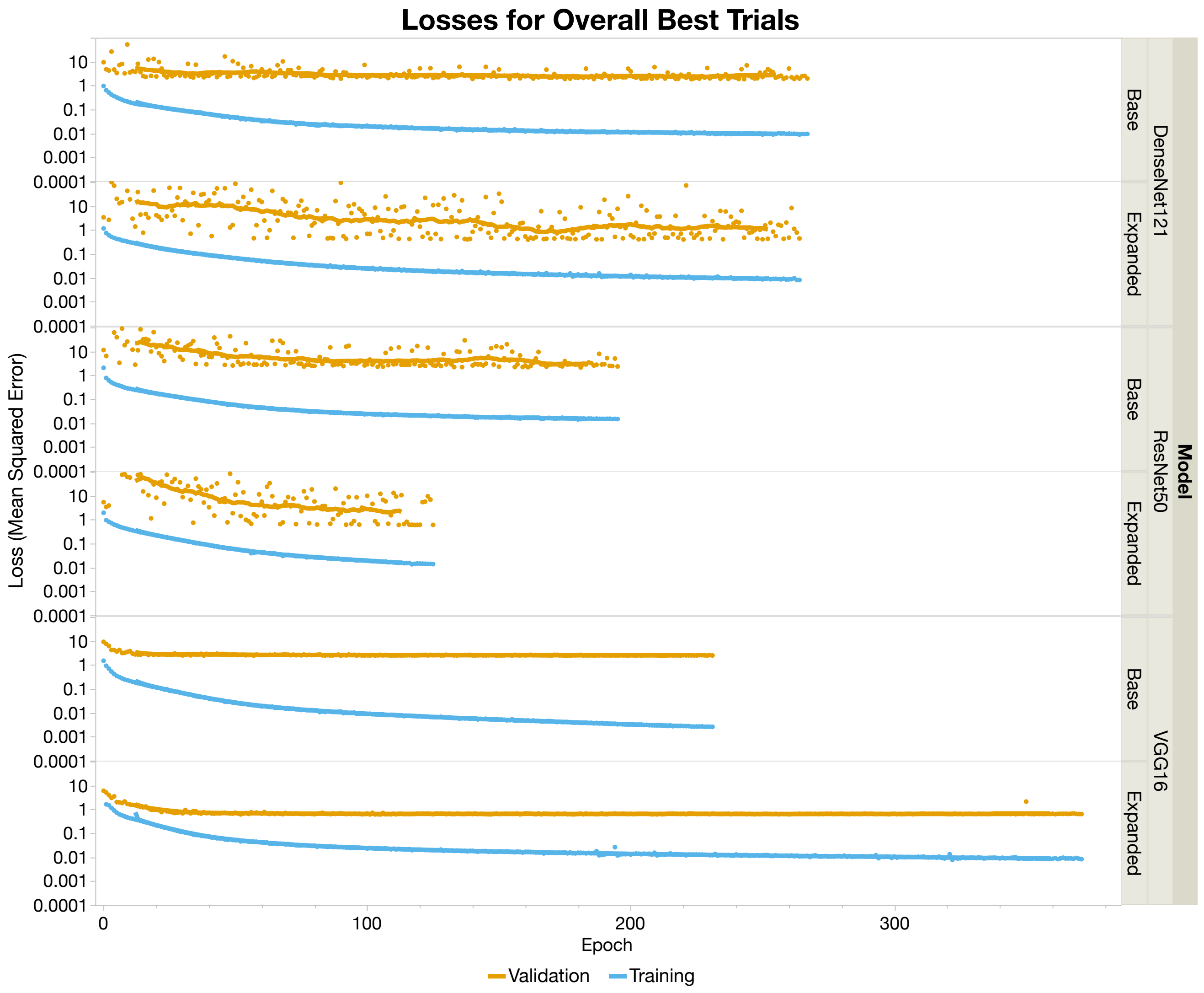


Figure 4: Training and Validation loss for the best trials, measured by total loss across parameters. Dots represent loss for specific epochs; matching lines represent a 25-epoch moving average.

is good enough for scientific computing is an ongoing endeavor, involving statistical and probabilistic comparison. Though certainly, a future effort in comparison could include training a network on both traditional DTMC approximate data and neuromorphic DTMC approximate and comparing performance when tested on Euler-Maruyama sampled data.

This report, however, has demonstrated that parameter recovery of stochastic processes from approximate data generated on neuromorphic hardware can be accomplished. Moreover, using off-the-shelf deep learning methods is effective for this particular application, and bespoke machine learning methods are not required. This helps provide confidence in the use of neuromorphic algorithms for numerical and scientific computing, and additionally yields a potential pathway for decreased energy consumption of synthetic/simulated training data. And while our results here used conventional GPUs for training, they do serve as a proof-of-concept and motivate the investigation of a fully-neuromorphic implementation.

6 ACKNOWLEDGMENTS

The authors would like to thank Walker Rickord for reviewing and providing feedback on a version of this manuscript.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525

This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE).

REFERENCES

[1] X. Chen, Z. Wei, M. Li, and P. Rocca, "A review of deep learning approaches for inverse scattering problems (invited review)," *Progress In Electromagnetics Research*, vol. 167, pp. 67–81, 2020.

[2] L. Pilozzi, F. A. Farrelly, G. Marcucci, and C. Conti, "Machine learning inverse problem for topological photonics," *Communications Physics*, vol. 1, no. 1, pp. 1–7, 2018.

[3] M. T. McCann, K. H. Jin, and M. Unser, "Convolutional neural networks for inverse problems in imaging: A review," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 85–95, 2017.

[4] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in neuroscience*, vol. 5, p. 73, 2011.

[5] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[6] S. Furber, "Large-scale neuromorphic computing systems," *Journal of neural engineering*, vol. 13, no. 5, p. 051001, 2016.

[7] J. M. Shainline, S. M. Buckley, R. P. Mirin, and S. W. Nam, "Superconducting optoelectronic circuits for neuromorphic computing," *Physical Review Applied*, vol. 7, no. 3, p. 034013, 2017.

[8] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[9] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[10] W. Severa, C. M. Vineyard, R. Dellana, S. J. Verzi, and J. B. Aimone, "Training deep neural networks for binary communication with the whetstone method," *Nature Machine Intelligence*, p. 1, 2019.

[11] M. Davies, "Benchmarks for progress in neuromorphic computing," *Nature Machine Intelligence*, vol. 1, no. 9, pp. 386–388, 2019.

[12] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Frontiers in neuroscience*, vol. 12, p. 774, 2018.

[13] S. Esser, P. Merolla, J. Arthur, A. Cassidy, R. Appuswamy, A. Andreopoulos, D. Berg, J. McKinstry, T. Melano, D. Barch *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing. 2016," *Preprint on ArXiv. http://arxiv.org/abs/1603.08270. Accessed*, vol. 27, 2016.

[14] C. M. Vineyard, R. Dellana, J. B. Aimone, F. Rothganger, and W. M. Severa, "Low-power deep learning inference using the spinnaker neuromorphic platform," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, 2019, pp. 1–7.

[15] K. E. Hamilton, C. D. Schuman, S. R. Young, N. Imam, and T. S. Humble, "Neural networks and graph algorithms with next-generation processors," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 1194–1203.

[16] J. B. Aimone, K. E. Hamilton, S. Mniszewski, L. Reeder, C. D. Schuman, and W. M. Severa, "Non-neural network applications for spiking neuromorphic hardware," in *Proceedings of the Third International Workshop on Post Moores Era Supercomputing*, 2018, pp. 24–26.

[17] W. Severa, O. Parekh, K. D. Carlson, C. D. James, and J. B. Aimone, "Spiking network algorithms for scientific computing," in *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 2016, pp. 1–8.

[18] O. Parekh, C. A. Phillips, C. D. James, and J. B. Aimone, "Constant-depth and subcubic-size threshold circuits for matrix multiplication," in *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*. ACM, 2018, pp. 67–76.

[19] J. B. Aimone, O. Parekh, C. A. Phillips, A. Pinar, W. Severa, and H. Xu, "Dynamic programming with spiking neural computing," in *Proceedings of the International Conference on Neuromorphic Systems*, ser. ICONS '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3354265.3354285

[20] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, 2021.

[21] S. G. Cardwell, C. Vineyard, W. Severa, F. S. Chance, F. Rothganger, F. Wang, S. Musuvathy, C. Teeter, and J. B. Aimone, "Truly heterogeneous hpc: Co-design to achieve what science needs from hpc," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, 2020, pp. 349–365.

[22] L. M. Ricciardi and L. Sacerdote, "The ornstein-uhlenbeck process as a model for neuronal activity," *Biological cybernetics*, vol. 35, no. 1, pp. 1–9, 1979.

[23] J. D. Smith and S. A. McKinley, "Assessing the impact of electrostatic drag on processive molecular motor transport," *Bulletin of mathematical biology*, vol. 80, no. 8, pp. 2088–2123, 2018.

[24] W. Wang, Y. Cai, Z. Ding, and Z. Gui, "A stochastic differential equation sis epidemic model incorporating ornstein–uhlenbeck process," *Physica A: Statistical Mechanics and its Applications*, vol. 509, pp. 921–936, 2018.

[25] E. Nicolato and E. Venardos, "Option pricing in stochastic volatility models of the ornstein-uhlenbeck type," *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics*, vol. 13, no. 4, pp. 445–466, 2003.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[28] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[29] K. Mosegaard and A. Tarantola, "Monte carlo sampling of solutions to inverse problems," *Journal of Geophysical Research: Solid Earth*, vol. 100, no. B7, pp. 12 431–12 447, 1995.

[30] K. Mosegaard and M. Sambridge, "Monte carlo analysis of inverse problems," *Inverse problems*, vol. 18, no. 3, p. R29, 2002.

[31] K. Mosegaard, A. Tarantola *et al.*, "Probabilistic approach to inverse problems," *International Geophysics Series*, vol. 81, no. A, pp. 237–268, 2002.

[32] F. Yaman, V. G. Yakhno, and R. Potthast, "A survey on inverse problems for applied sciences," *Mathematical problems in engineering*, vol. 2013, 2013.

[33] S. U. Khan, S. Yang, L. Wang, and L. Liu, "A modified particle swarm optimization algorithm for global optimizations of inverse problems," *IEEE Transactions on Magnetics*, vol. 52, no. 3, pp. 1–4, 2015.

[34] W. Severa, R. Lehoucq, O. Parekh, and J. B. Aimone, "Spiking neural algorithms for markov process random walk," in *International Joint Conference on Neural Networks 2018*. IEEE, 2018.

[35] J. D. Smith, W. Severa, A. J. Hill, L. Reeder, B. Franke, R. B. Lehoucq, O. D. Parekh, and J. B. Aimone, "Solving a steady-state pde using spiking networks and neuromorphic hardware," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–8.

[36] J. D. Smith, A. J. Hill, L. E. Reeder, B. C. Franke, R. B. Lehoucq, O. Parekh, W. Severa, and J. B. Aimone, "Neuromorphic scaling advantages for energy-efficient random walk computations," *Nature Electronics*, vol. 5, no. 2, pp. 102–112, 2022.

[37] J. B. Aimone, R. Lehoucq, W. Severa, and J. D. Smith, "Assessing a neuromorphic platform for use in scientific stochastic sampling," in *2021 International Conference on Rebooting Computing (ICRC)*. IEEE, 2021, pp. 64–73.

[38] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[39] A. V. Skorokhod, *Studies in the theory of random processes*. Courier Dover Publications, 1982, vol. 7021.