(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
06 June 2024 (06.06.2024)

WIPO | PCT

(10) International Publication Number
WO 2024/118915 A1

(51) International Patent Classification:
G05B 13/02 (2006.01) *G06N 7/06* (2006.01)
G05B 13/04 (2006.01) *G06N 20/00* (2019.01)
G06N 7/04 (2006.01)

(21) International Application Number:
PCT/US2023/081816

(22) International Filing Date:
30 November 2023 (30.11.2023)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
63/385,891 02 December 2022 (02.12.2022) US
63/478,710 06 January 2023 (06.01.2023) US
63/483,856 08 February 2023 (08.02.2023) US

(71) Applicant: **NORMAL COMPUTING CORPORATION** [US/US]; 16 W Hubbard St., #10, Chicago, IL 60654 (US).

(72) Inventors: **COLES, Patrick**; 3829 Oasis Springs Rd NE, Rio Rancho, NM 87144 (US). **SZCZEPANSKI, Collin**; 18 Rutgers Lane, Princeton, NJ 08540 (US). **DONATELLA, Kaelan**; 50 rue de la goutte d'or, 75018 Paris (FR). **MELANSON, Denis**; 202-80 King William Street, Hamilton, ON L8R 0A1 (CA). **SBAHI, Faris**; 154 W 27th St Rm 4E, New York, NY 10001 (US). **MARTINEZ, Antonio**; 444 W Arlington Pl #2, Chicago, IL 60614 (US).

(74) Agent: **COLICE, Christopher, Max** et al.; Smith Baluch LLP, 376 Boylston St., Suite 401, Boston, MA 02116 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO,

(54) **Title:** THERMODYNAMIC ARTIFICIAL INTELLIGENCE FOR GENERATIVE DIFFUSION MODELS AND BAYESIAN DEEP LEARNING

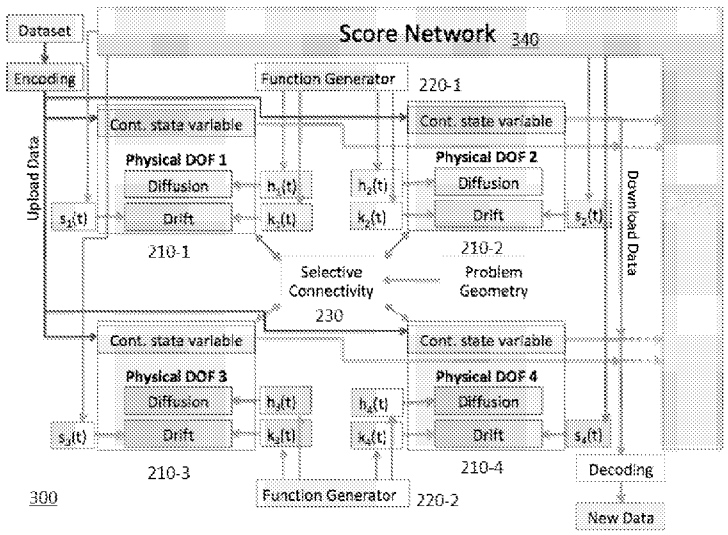

Figure 3

(57) **Abstract:** A physics-based system performs generative modeling on a given dataset by turning the diffusion process in diffusion models into a physical process. Electrical circuits provide an exemplary implementation, where each unit cell (an electrical circuit in a network of electrical circuits) is composed of resistors, a stochastic noise source (such as a thermal or shot noise source), programmable voltage sources, and a capacitor whose charge encodes the state variable. These unit cells can be capacitively coupled with a connectivity that matches the problem geometry. A score network, which provides predictions for score values, can be implemented on a digital, analog, or hybrid digital-arialog device. A detailed construction for an analog score network is provided in the form of a physical system that evolves over time, simultaneously with the diffusion process. The analog score network allows score values to be provided continuously to the reverse diffusion process without latency, and also allows for efficient evaluation of the loss function when coupled to the forward diffusion process.

WO 2024/118915 A1

[Continued on next page]

RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

Thermodynamic Artificial Intelligence for Generative Diffusion Models and Bayesian Deep Learning

Cross-Reference to Related Applications

This application claims the priority benefit, under 35 U.S.C. 119(e), of U.S. Application No. 63/483,856, filed February 8, 2023, and entitled "Thermodynamic Artificial Intelligence System for Bayesian Deep Learning," U.S. Application No. 63/478,710, filed January 6, 2023, and entitled "Thermodynamic Artificial Intelligence System for Generative Diffusion Models," and U.S. Application No. 63/385,891, filed December 2, 2022, and entitled "Thermodynamic Artificial Intelligence for Generative Diffusion Models." Each of these applications is incorporated herein by reference in its entirety for all purposes.

Background

1 Generative Modeling

Recently, generative modeling (GM) has emerged as one application of machine learning (ML) and artificial intelligence (AI). For example, the text-to-image application of GM has captured the imagination of users, allowing them to generate their own artwork simply by typing in words. Also, generating seemingly realistic (but ultimately fake) images of human faces demonstrates the power of GM. Generation of text, audio, computer code, and molecular structures are additional applications of GM.

A generative model uses a probabilistic framework and describes how a dataset is generated in terms of a probabilistic model. One can then sample from this model in order to generate new data. There are several approaches to GM. Generative Adversarial Networks (GANs) were popular in the early days of GM. GANs employ two neural networks acting in an adversarial setting, where one network tries to discriminate the output of the other from real data samples. More recently, Diffusion Models have been introduced for GM and typically have superior performance over GANs.

2 Diffusion Models

A class of physics-inspired models, known as Diffusion Models (DMs), has recently revolutionized the field of GM. DMs have become competitive with GANs and other GM methods in image synthesis, video generation, and molecule design.

2.1 Score SDE

A unified framework for diffusion models based on stochastic differential equations (SDEs), called the Score SDE approach, follows four steps:

1. Generate training data by evolving under the Forward SDE, which adds noise to data from the dataset of interest.

2. Use this data to train a neural network (the "score network") in order to match the score (the gradient of the logarithm of the probability) associated with the distribution at each noise level.

3. Produce a sample from the noisy distribution, i.e., the distribution associated with the final time point of Step 1.

4. Evolve this sample under the Reverse SDE (which is defined using the trained score network) to generate a novel datapoint.

Once Step 2 is done, Steps 3 and 4 can be repeated many times to generate many novel datapoints. FIG. ⁞ shows how Steps 3 and 4 can be repeated in the context of image processing.

In general, the forward SDE (in Step 1 above) takes the form

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + \mathbf{G}(t)\mathrm{d}\mathbf{w} \tag{1}$$

where $\mathbf{x}$ is the data vector, $\mathrm{d}t$ is a positive timestep, $\mathrm{d}\mathbf{w}$ denotes a Gaussian-distributed noise term associated with Brownian motion, $\mathbf{f}(\mathbf{x}, t)$ is a function that determines the drift, and $\mathbf{G}(t)$ acts as the diffusion tensor. We assume that $\mathbf{G}(t)$ depends only on t and not on $\mathbf{x}$.

The Reverse SDE (in Step 4 above) reads:

$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \mathbf{G}(t)\mathbf{G}(t)^{\intercal}\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right]\mathrm{d}t + \mathbf{G}(t)\mathrm{d}\overline{\mathbf{w}}, \tag{2}$$

where $\mathrm{d}t$ is a negative timestep here and $\mathrm{d}\overline{\mathbf{w}}$ represents a Brownian motion in reverse time (i.e., with time running backwards). In this equation, the score is usually approximated with a trainable neural network, i.e., $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \approx \mathbf{s}_\theta(\mathbf{x}, t)$.

This general framework encompasses several special cases of interest. The Variance Preserving (VP) process is the continuous version of the discrete Markov chain often used in Denoising Diffusion Probabilistic Models (DDPMs). The SDEs for the VP process are:

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\mathbf{w} \quad \text{(Forward)} \tag{3}$$

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\mathrm{d}t - \beta(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})dt + \sqrt{\beta(t)}\mathrm{d}\overline{\mathbf{w}} \quad \text{(Reverse)} \tag{4}$$

The Variance Exploding (VE) process is the continuous version of the discrete Markov chain used in Score Matching with Langevin Dynamics (SMLD). The SDEs for the VE process are:

$$\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}\mathrm{d}\mathbf{w} \quad \text{(Forward)} \tag{5}$$

$$\mathrm{d}\mathbf{x} = -\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\mathrm{d}t + \sqrt{\frac{\mathrm{d}\sigma^2(t)}{\mathrm{d}t}}\mathrm{d}\overline{\mathbf{w}} \quad \text{(Reverse)} \tag{6}$$

One more special case is a unification of the VP and VE processes:

$$\mathrm{d}\mathbf{x} = f(t)\mathbf{x}\mathrm{d}t + g(t)\mathrm{d}\mathbf{w} \quad \text{(Forward)} \tag{7}$$

$$\mathrm{d}\mathbf{x} = f(t)\mathbf{x}\mathrm{d}t - g(t)^2\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\mathrm{d}t + g(t)\mathrm{d}\overline{\mathbf{w}} \quad \text{(Reverse)} \tag{8}$$

for some time-dependent functions $f(t)$ and $g(t)$. The VP and VE processes are special cases of these equations.

2.2 Score matching

One subroutine of DMs (see Step 2 in the above list) is training the score network such that its output $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, t)$ matches the exact score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. Ideally, the score should match at every noise level (i.e., at every time t). To accomplish this, the following loss function can be reduced or minimized over variational parameters $\boldsymbol{\theta}$:

$$\mathcal{L}_1(\boldsymbol{\theta}) = \mathbb{E}_t \left\{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)} \left[\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t)|\mathbf{x}(0)) \|_2^2 \right] \right\}. \tag{9}$$

This loss function is a weighted average over input data $\mathbf{x}(0)$ and noised input data $\mathbf{x}(t)|\mathbf{x}(0)$ (conditioned on the input data $\mathbf{x}(0)$) taken at different times t that are uniformly drawn from the time interval $[0, T]$ of the forward SDE. This loss function uses knowledge of the form of the conditional distributions $p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))$ associated with the forward process.

In some cases, the form of the conditional distributions may be unknown. In these cases, an alternative loss function can be useful. Indeed, an alternative loss function based on the trace of the Hessian of the log probability can be written as

$$\mathcal{L}_2(\boldsymbol{\theta}) = \mathbb{E}_t \left\{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)} \left[\frac{1}{2} \| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) \|_2^2 + \mathrm{Tr}[\nabla_{\mathbf{x}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)] \right] \right\}. \tag{10}$$

Both loss functions, $\mathcal{L}_1$ and $\mathcal{L}_2$, are useful for score matching in diffusion models. In practice, these loss functions would be estimated via finite sampling. Hence, an unbiased estimator for the loss function (or its gradient) can be constructed using a finite number of samples to estimate the expectation values in the loss function.

2.3 Score networks

A score network is a trainable neural network whose output is denoted by the function $s_\theta(\mathbf{x}, t)$. It aims to approximate the true score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, for each time t and each input vector $\mathbf{x}$.

On the one hand, the score network should be expressive enough to model complicated functions. After all, the distributions $p_t(\mathbf{x})$ can be highly complex (e.g., multimodal) and hence so can their associated score functions. Therefore, typically a score network should have many parameters to be expressive enough. The output of the score network $s_\theta(\mathbf{x}, t)$ can be a composition of affine transformations and non-linear activation functions. This is similar to how standard neural networks are constructed. Neural networks are good at approximating a wide variety of functions.

On the other hand, inductive bias can improve the performance of the score network, including its trainability and generalization performance. In practice, inductive bias often corresponds to accounting for the problem geometry when contructing the neural network. For example, the problem geometry may correspond to a grid (e.g., in 1D, 2D, or 3D) or graph. Implementing such geometrical inductive biases is useful in fields such as molecule synthesis and material design, where including symmetries as inductive biases substantially improves the performance of diffusion models.

Therefore, the construction of the score network can depend on the nature of the problem. For example, when considering 2D images, it is common to employ a so-called U-Net for the score network. The U-Net was originally introduced for medical image segmentation, and it has a structure similar to a convolutional neural network. Such networks account for spatial locality and problem geometry.

In summary, score networks should be constructed in such a way to be both expressive as well as accounting for problem geometry.

2.4 Latent and Spectral Diffusion Models

Some extensions of DMs allow for some pre- and post-processing of the data. For example, Latent Diffusion Models (LDMs) first pass the data through a trained autoencoder that transforms the data from data space to latent space, then act with the DM in the latent space, and then reverse the autoencoder to go back to the data space. This reduces the computational difficulty for the DM since the latent space is lower dimensional.

Similarly, Spectral Diffusion Models (SDMs) first transform spatial data into the spectral domain prior to acting with the DM and then transform back in the end, which allows the noise process to be correlated in the spatial domain.

As a particularly interesting example of this, generative approaches for images based on the heat equation and blurring diffusion can be thought of as employing uncorrelated noise in the frequency domain. With this interpretation, the drift or diffusion terms should depend on frequency. In other words, the functions $\mathbf{f}$ or $\mathbf{G}$ in Eq. (⋮) may have a non-trivial dependencies on $\mathbf{x}$.

With the success of these methods involving data transformations, it then makes sense to modify the general protocol for Score SDE. Namely, the Score SDE protocol can be supplemented by additional steps: (0) Pre-process the data with an encoding process, (1) - (4) Perform the Score SDE protocol given above on the encoded data, (5) Map the encoded space back to the data space.

2.5 Probability Flow ODE

Every SDE has an associated Ordinary Differential Equation (ODE) flow for its underlying probability distribution. For the score SDE, the trajectories described by $\mathbf{x}(t)$ obey the probability distribution $p_t(\mathbf{x})$ whose evolution is governed by:

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\sum_{i=1}^{N} \frac{\partial}{\partial x_i}[\tilde{\mathbf{f}}_i(\mathbf{x}, t)p_t(\mathbf{x})], \tag{11}$$

with

$$\tilde{\mathbf{f}}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}, t) - \frac{1}{2}\mathbf{G}(t)\mathbf{G}(t)^{\intercal}\nabla_{\mathbf{x}} \log p_t(\mathbf{x}). \tag{12}$$

This can be used to obtain an ODE on the data $\mathbf{x}$ that reads

$$\mathrm{d}\mathbf{x} = \tilde{\mathbf{f}}(\mathbf{x}, t)\mathrm{d}t. \tag{13}$$

This equation can be propagated forward and backward in time to match the forward and reverse processes. In practice, the probabilistic flow ODE may be used only for the reverse process, i.e., for sample generation since it involves the score function. Hence probabilistic flow ODE cannot be used for the training procedure. In some cases, using the probabilistic flow ODE for the reverse process improves sample generation.

2.6 Computational bottlenecks

Currently, the above framework for DMs is implemented on standard digital hardware. This leads to several computational difficulties:

1. Large amount of training data for training the score network;

2. Large complexity for training and evaluating the score network;

3. Long times for numerically simulating the dynamics in the Forward and Reverse SDEs or ODEs; and

4. Potential instability of SDEs/ODEs due to large dimensionality and stiffness, leading to impractical extremely small time steps

The first two difficulties above largely stem from a lack of inductive bias in the problem setup. Inductive bias refers to prior knowledge being inputted into the structure of the model, for example, knowledge of symmetries in the data. Having a strong inductive bias can reduce the training data requirements as well as improve the speed of training of the model.

The third difficulty refers to the challenge of using digital hardware to simulate time dynamics. This includes both the challenge of digitally generating Gaussian randomness (i.e., the d**w** term in the SDE) and the challenge of numerical time integration via discretizing the time dynamics. Finally, the fourth difficulty refers to the fact that there are many cases in which numerically integrating SDEs and ODEs may lead to instabilities due to the structure of the equations, particularly for SDEs when the number of dimensions is large. Another issue is known as stiffness, which leads to smaller time steps and can make numerical integration impractical.

Summary

Diffusion models are physics-inspired since diffusion is a physical process. This raises the question of whether a physical process could mimic the mathematics used in, say, the Score SDE protocol above.

Nature can perform computations that otherwise would lead to large computational overhead with a standard digital computer. This paradigm of computation through nature is the motivation for both analog computers and quantum computers.

Since diffusion is a thermodynamic process involving randomness, a nature-based computer for this application can be thermodynamic, involving physical randomness (e.g., due to heat fluctuations). A thermodynamic system can address the computational bottlenecks encountered when trying to solve diffusion models with standard digital computers. In particular, a physical system can address the inductive bias issue as well as the time dynamics simulation issue raised above.

Examples of the inventive technology aims to remove the computational bottlenecks encountered by generative diffusion models run on purely digital hardware. These examples exploit the insight that diffusion is a natural process that occurs in many physical systems, including electrical circuits and continuous-variable optical systems.

One example of the inventive technology can be implemented as a system for executing a generative diffusion model on a dataset. This system includes a physical system with several degrees of freedom, each of which is associated with a corresponding feature of the dataset. Each degree of freedom has a continuous state variable that evolves according to a corresponding differential equation having a tunable diffusion term and a tunable drift term. The degrees of freedom can be physically coupled to each other according to a problem geometry associated with the dataset.

The system may also include a processor, operably coupled to the physical system, to reduce the entropy of the continuous state variables so as to produce an output of the generative diffusion model. For example, the processor can reduce the entropy of the continuous state variables by reading off the continuous state variables and altering at least one of the tunable drift terms and/or by executing a score network. The processor may be configured to be trained by an optimization routine that reduces a loss function. The physical system may be configured to assist with estimating the loss function, and/or the processor may include analog circuitry configured to assist with estimating the loss function.

Estimating the loss function may involve simultaneous time evolution of the physical system and of the analog circuitry in the processor. The loss function may quantify a degree of score matching.

The system can also include a function generator that is operably coupled to the physical system. In operation, the function generator multiplies the tunable drift terms and tunable diffusion terms by arbitrary time-dependent functions so as to modify the differential equations governing evolution of the continuous state variables. And the system can include a digital device that is operably coupled to the physical system. In operation, the digital device uploads the dataset to the degrees of freedom and downloads new data corresponding to measurements of values of the continuous state variables after evolution of the continuous state variables.

The physical system can include a network of electrical circuits, each of which provides a corresponding degree of freedom and includes a capacitor having a charge encoding the continuous state variable; a stochastic noise source, in series with the capacitor, to generate the tunable diffusion term; and a resistor, in series with the capacitor, to generate the tunable drift term. Each electrical circuit can also include first and second tunable voltage sources operably coupled to the stochastic noise source and the resistor, respectively. The first tunable voltage source tunes the tunable diffusion term of the differential equation, and the second tunable voltage source tunes the tunable drift term of the differential equation. The stochastic noise source may be a thermal noise source and/or a shot noise source.

Alternatively, each electrical circuit in the network of electrical circuits could include a variable resistor, in parallel with the capacitor and having a variable resistance controlled by a tunable voltage source, to tune the tunable drift term of the differential equation; and an amplifier, operably coupled to the stochastic noise source, to amplify an output of the stochastic noise source so as to tune the tunable diffusion term of the differential equation. The amplifier may have a variable gain determined by an additional variable resistor whose resistance is controlled by an additional tunable voltage source.

The physical system can also include switches coupling the electrical circuits in the network of electrical circuits according to a problem geometry associated with the dataset. The physical system may also include switches configured to be actuated between settings for a forward diffusion process and settings for a reverse diffusion process.

The overall system can include a processor, operably coupled to the network of electrical circuits, to reduce entropy of the continuous state variables so as to produce an output of the generative diffusion model. Again, the processor can reduce the entropy of the continuous state variables by reading off the continuous state variables and altering at least one of the tunable drift terms. The processor may include analog circuitry configured to evolve over time simultaneously with the physical system. The analog circuitry may be configured to continuously output predictions for score values as analog signals to be used as inputs to the physical system. The score values can be produced by physically modeling, with trainable physical devices, partial derivatives of the score with respect to time and with respect to the continuous state variables. The trainable physical devices used to model the partial derivatives may be artificial neural networks. The processor may include an integrator circuit, operably coupled to the trainable physical devices, to produce the score values by integrating, over time, outputs of the trainable physical devices. The processor may further include a field-programmable gate array (FPGA).

The analog circuitry in the processor may include a network of electrical circuits, each electrical circuit in the network of electrical circuits providing a corresponding degree of freedom and comprising: a capacitor; a resistor in series with the capacitor; and a voltage source in series with the capacitor. At least one electrical circuit in the network of electrical circuits may be capacitively coupled to at least one other electrical circuit in the network of electrical circuits according to a problem geometry associated with the dataset. The voltage source may be a multi-layer neural network configured to take the continuous state variable as input and to output a voltage value for each electrical circuit in the network of electrical circuits.

All combinations of the foregoing concepts and additional concepts discussed in greater detail below (provided such concepts are not mutually inconsistent) are contemplated as being part of the inventive subject matter disclosed herein. In particular, all combinations of claimed subject matter appearing at the end of this disclosure are contemplated as being part of the inventive subject matter disclosed herein. The terminology explicitly employed herein that also may appear in any disclosure incorporated by reference should be accorded a meaning most consistent with the particular concepts disclosed herein.

Brief Descriptions of the Drawings

The skilled artisan will understand that the drawings primarily are for illustrative purposes and are not intended to limit the scope of the inventive subject matter described herein. The drawings are not necessarily to scale; in some instances, various aspects of the inventive subject matter disclosed herein may be shown exaggerated or enlarged in the drawings to facilitate an understanding of different features. In the drawings, like reference characters generally refer to like features (e.g., functionally similar and/or structurally similar components).

FIG. 1 illustrates a diffusion model applied to an image. The forward diffusion process, or forward process, adds noise to each pixel, while the reverse diffusion process, or reverse process, removes noise from the image to produce a new data point.

FIG. 2 is a schematic diagram of our thermodynamic AI device for simulating the forward process of generative diffusion models. The physical system is composed of multiple degrees of freedom (DOFs). For simplicity we show four DOFs here, although in general the number of DOFs matches the dimensionality of the data, i.e., the number of features in the data. Each DOF has a continuous state variable, and that variable evolves according to a differential equation that, in general, could have both a diffusion and drift term. A function generator is capable of multiplying these diffusion and drift terms by arbitrary time-dependent functions, respectively $h_j(t)$ and $k_j(t)$, for the jth DOF. The problem geometry, associated with a given dataset, can be uploaded onto the device by selectively connecting the various DOFs, which mathematically couples the differential equations of the various DOFs. After some encoding, a datapoint from the dataset of interest can be uploaded to the device by initializing the values of the continuous state variables to be the corresponding feature values of the datapoint. Similarly, data can be downloaded (and decoded) from the device by measuring the values of the continuous state variables after some time evolution.

FIG. 3 is a schematic diagram of our thermodynamic AI device for simulating the reverse process of generative diffusion models. In addition to all of the device components that are present in the forward process, the reverse process uses a trained score network. The inputs to the score network are the values of the continuous state variables at some time t, and the output is the value of the score. The jth component, $s_j(t)$, of the score gets added as a drift term in the evolution of the jth DOF. The score network acts as a Maxwell's demon, which continuously monitors the physical system and appropriately adapts the drift term in order to reduce the physical system's entropy.

FIG. 4 shows a circuit diagram for the unit cell, the building block of our analog device. The voltage functions $k(t)$ and $h(t)$ are chosen by the user to allow for different drift and diffusion coefficients and are multiplied by the intrinsic circuit voltages using voltage mixers (circle with cross). We remark that the circuit shown here is for solving the SDE formulation of diffusion models. However, to solve the ODE formulation, we can remove the thermal noise source w shown here or, in other words, operate in a regime where this thermal noise is negligible.

FIG. 5 is a circuit diagram of one unit cell of the forward process using variable resistors to control the drift and diffusion terms.

FIG. 6 shows two unit cells coupled to each other via a coupling capacitor C_{12}. This coupling forms the basis for connecting unit cells in our analog device, in general.

FIG. 7 is a circuit diagram of two capacitively coupled unit cells using variable resistors. The images above the main circuit diagram are definitions of shorthand circuit symbols used in the main diagram.

FIG. 8 shows a simplified version of the circuit used in our two unit cells experimentally implemented as a heat engine. The plot shows that the voltages across the capacitors in the unit cells were highly correlated with a large coupling capacitance. This suggests that a coupling capacitor effectively correlates the random walks (i.e., the noise processes) in the unit cells. This spatial correlation is desirable to engineer the inductive bias of the model.

FIG. 9 illustrates four possible problem geometries, although there are other possibilities. DNA sequences have a 1D geometry, images have a 2D geometry, solutions to partial differential equations (PDEs) in real space (such as fluid flow) often have a 3D geometry, and molecular structures have some graph connectivity and hence follow the geometry of a graph.

FIG. 10 illustrates mapping the connectivity matrix onto the hardware. Each off-diagonal element of the connectivity matrix is translated into the state of a switch in a wire connecting two unit cells. For simplicity, the capacitors in series with the switches are not shown. For a Hidden Layer Network, the switches are placed in series with the resistive bridges that bridge the unit cells. For the Prior Weight Diffuser and the Posterior Weight Diffuser, the switches are placed in series with the capacitive bridges that bridge the unit cells.

FIG. 11 shows a comparison of the typical scenario for Maxwell's Demon (left panel) with our scenario (right panel). In our scenario, the voltage across a capacitor in an electrical circuit plays the role of the dynamical variable, analogous to the positions of the gas particles in the left panel. Even if these voltages start in a high entropy state (illustrated by different colors of the different capacitors), they can evolve over time towards a low entropy state (illustrated by a single color for the different capacitors). This entropy reduction is facilitated by an intelligent observer (also known as a Maxwell's Demon) which continuously observes the state of the system and adapts the applied voltage in each circuit appropriately.

FIG. 12 is a schematic illustration of how the score network, which is stored and executed on a digital device such as a central processing unit (CPU) or a field-programmable gate array (FPGA), interacts with the entire set of analog unit cells via Analog-to-Digital converters (ADCs) and Digital-to-Analog converters (DACs).

FIG. 13A shows coupling the digital score network (SN) to the analog unit cell, for solving the Reverse SDE, where the output of the SN is multiplied by a function $g(t)^2$ in an analog fashion.

FIG. 13B shows coupling the digital SN to the analog unit cell, for solving the Reverse SDE, where the output of the score network is multiplied on a digital device.

FIG. 14 is a flow chart for our Thermodynamic AI system whenever the score network has already been pre-trained.

FIG. 15 is a flow chart for our Thermodynamic AI system when the score network is untrained.

FIG. 16 is a schematic circuit diagram for the score device. This diagram represents the process used to obtain score values during the evolution of the reverse process.

FIG. 17 shows circuit diagrams for a voltage adder (left) and a voltage integrator (right).

FIG. 18 shows a simple circuit that provides a subroutine that computes the norm squared of a voltage vector $v = \{v_1, v_2, ..., v_d\}$.

FIG. 19 is a circuit diagram for a root-mean-square (RMS) converter based on the thermal method.

FIG. 20 is a schematic illustration of a training process for the analog score device.

FIG. 21 is a flow chart for our thermodynamic AI system when the score network is an analog device, and this analog device is used for evaluating the loss function during training and for interfacing with the reverse process after training.

FIG. 22A shows one unit cell of an analog score device.

FIG. 22B shows two unit cells of the analog score device, capacitively coupled together.

FIG. 23 shows a layered analog neural network to evaluate functions $\mathbf{r}^{(i)}$ (also applies to evaluating $\mathbf{q}$). The $N+1$ dimensional input $(\mathbf{v}(t), t)$ is fed into a parametrized layer A_1, that is detailed in Fig. 24. After each linear layer, the output is then fed into a diode followed by a resistor whose current output is a nonlinear function f_d of the voltage. Here, the last nonlinear layer is shown and is denoted by B_K.

FIG. 24 shows the first resistive layer of the analog neural network. (All resistive layers take the same generic form, possibly with different hyperparameters.) Each input voltage is copied M_1 times, so that they can be fed into $M_1 \times d + 1$ resistors with inverse values being the entries of the $\mathbf{A}_1$ matrix $(A_1)_{j,k}$. This circuit produces a weighted average of the input voltages, stored in each $A_1[\mathbf{v}(t), t]_j$, which is then fed to the nonlinear layer. Although not shown in the figure, amplifiers can be added at each layer to amplify the output voltage, to prevent the voltage from decaying from layer to layer.

FIG. 25 is a schematic of a basic voltage follower. According to the two "golden rules" of operational amplifiers, we have that $V_{\text{in}} = V_{\text{out}}$, while simultaneously we have that no current enters either op amp input.

FIG. 26 is a schematic of the circuit to compute the dot product between the input voltages and the j^{th} row of $\mathbf{A}_1$.

FIG. 27 shows four cases of communication between the reverse diffusion process and the score device is shown (top) and simplified circuit notations (bottom) for a single unit cell ($N = 1$). These cases include when the score device is analog or hybrid digital-analog, combined with the possibility that the unit cell is either based on voltage mixers or on variable resistors.

FIG. 28 shows a circuit diagram for a universal unit cell that can be used for both the forward and reverse processes. The thick slashes through certain wires indicate that a switch is added in series with the wire. When all of these switches are open (closed), the unit cell corresponds to that used in the forward (reverse) process. This allows the same device to be used for both the forward and reverse processes.

FIG. 29 illustrates how a Bayesian neural network (BNN) differs from a standard neural network, showing that the weights in a BNN have some probability distribution.

FIG. 30 illustrates a multimodal distribution. The posterior distribution of the weights in a BNN is typically multimodal.

FIG. 31 shows how the four components or subsystems of a BNN interact and feed signals to each other.

FIG. 32 shows a unit cell for the hidden layer network (HLN), including a capacitor C_j, a resistor R_j, a non-linear element (NLE) in parallel with the capacitor, and a voltage source α_j.

FIG. 33 shows two units cell for the HLN coupled together via a resistive bridge.

FIG. 34 shows an analog augmented neural ODE, which can be used for the HLN.

FIG. 35 shows a unit cell for the prior weight diffuser.

FIG. 36 shows two unit cells, for the prior weight diffuser, coupled together via a capacitive bridge.

FIG. 37A shows a unit cell for the posterior weight diffuser for the cases where the posterior drift network (PDN) is an analog device. The diagram illustrates a feedback loop, where the voltage across the capacitor is measured and fed as input to the PDN, which then applies an appropriate drift voltage to the unit cell.

FIG. 37B shows a unit cell for the posterior weight diffuser for the cases where the PDN is a digital device, such as an FPGA.

FIG. 38 illustrates outputting weights from the weight diffuser device to the HLN device for two unit cells of the weight diffuser. This concept applies to an arbitrary number of unit cells.

FIG. 39 illustrates inputting weights from the weight diffuser device into the HLN device for two unit cells of the HLN. This concept applies to an arbitrary number of unit cells.

FIG. 40 is a circuit diagram illustrating that the same device can be used for the prior and posterior weight diffusers. The thick lines represent switches that allow one to toggle between the two different diffusers. When the switches are open (closed), the device corresponds to the prior (posterior) weight diffuser.

FIG. 41 illustrates a feedback process between the posterior weight diffuser and the posterior drift network for the case where the posterior drift network is a digital device, such as an FPGA or CPU.

FIG. 42 depicts a layered analog neural network configured to evaluate functions $\mathbf{r}^{(\phi)}$, $\mathbf{q}^{(\phi)}$, or $\mathbf{s}^{(\phi)}$, in the context of the PDN. The $W+1$ dimensional input $(\mathbf{w}(t), t)$ is fed into a parameterized layer, corresponding to an affine matrix A_1, that is detailed in Fig. 43. After each linear layer, the output is then fed into a NLE followed by a resistor whose current output is a nonlinear function of the voltage. Here, the last nonlinear layer is denoted by B_K.

FIG. 43 shows the first resistive layer of the analog neural network. (The resistive layers take the same generic form, possibly with different hyperparameters.) Each input voltage is copied M_1 times, so that they can be fed into $M_1 \times W + 1$ resistors with inverse values being the entries of the $\mathbf{A}_1$ matrix $(A_1)_{j,k}$. This circuit produces a weighted average of the input voltages, stored in each $A_1[\mathbf{w}(t), t]_j$, which is then fed to the nonlinear layer. Although not shown in the figure, amplifiers can be added at each layer to amplify the output voltage, to prevent the voltage from decaying from layer to layer.

FIG. 44 is a schematic diagram of the circuit used to integrate the total derivative for the drift, in order to output a drift value from the PDN.

FIG. 45 is a circuit diagram for the estimation of the λ term in the loss function, employing digital time integration.

FIG. 46 is a circuit diagram for the estimation of the λ term in the loss function, employing analog time integration.

FIG. 47 shows a unit cell for an analog neural ODE employing a voltage mixer. This unit cell can allow for reversing the direction of time based on the choice of function $k(t)$, e.g., by choosing $k(t) \to -k(1-t)$.

FIG. 48 shows two unit cells for the adjoint device. Each cell has a resistor and a capacitor, and the cells are coupled via a resistive bridge. The adjoint device evolves the adjoint variable $\mathbf{a}$ over time, where this variable can be encoded in the voltages across the capacitors in the device. This is in the context of the adjoint sensitivity method for computing gradients of neural ODEs.

FIG. 49 shows a circuit diagram for an integrator circuit.

FIG. 50 illustrates an analog latent ODE for fitting and extrapolating time-series data.

FIG. 51 shows a unit cell for an analog neural SDE processor.

FIG. 52 illustrates various algorithms unified under a single mathematical framework of thermodynamic AI algorithms.

FIG. 53 is a circuit diagram of a physical realization of an s-mode comprising a noisy resistor and a capacitor.

FIGS. 54A and 54B are circuit diagrams of physical realizations of coupling between s-modes using a coupling resistor and a coupling capacitor, respectively.

FIG. 55 illustrates a force-based approach to constructing a Maxwell's Demon device.

Detailed Description

3 Generic Physical Thermodynamic Artificial Intelligence (AI) Devices

FIGS. 2 and 3 are schematic diagrams showing a generic physical architecture of a thermodynamic AI device. FIG. 2 illustrates the thermodynamic AI device 200 performing the forward process, and FIG. 3 illustrates the thermodynamic AI device 300 performing the reverse process. This architecture can be implemented in electrical circuits, as described below, or in other physical systems, such as continuous-variable optics systems.

As shown in FIG. 2, the physical system 200 has multiple degrees of freedom (DOFs) 210-1 through 210-4. The number of DOFs 210 matches the dimensionality of the data, i.e., the number of features in the input dataset 201. Each DOF 210 has a continuous state variable, and that variable evolves according to a differential equation that, in general, could have both a diffusion and drift term. Function generators 220-1 and 220-2 can multiply these diffusion and drift terms by arbitrary time-dependent functions. The problem geometry 203, associated with a given dataset, can be uploaded onto the device by selectively connecting the various DOFs 210 with switches 230. This mathematically couples the differential equations of the various DOFs 210. After some encoding, a datapoint from the dataset 201 of interest can be uploaded to the device by initializing the values of the continuous state variables to be the corresponding feature values of the datapoint. Similarly, new data 209 can be downloaded (and decoded) from the device by measuring the values of the continuous state variables after some time evolution.

In addition, the reverse process uses a trained score network 340, as in FIG. 3. The inputs to the score network are the values of the continuous state variables at some time t, and the output is the value of the score. The jth component of the score gets added as a drift term in the evolution of the jth DOF. The score network 340 acts as a Maxwell's Demon that continuously monitors the physical system and appropriately adapts the drift term in order to reduce the physical system's entropy.

4 Electrical Circuit Thermodynamic AI Device

A thermodynamic AI device can be implemented as a hybrid analog-digital system that is thermodynamic in nature. This analog system generates diffusion via a thermal noise system, such as an electrical resistor. As noted above, Gaussian randomness is costly to generate digitally and hence is better generated with an analog system. Moreover, time dynamics are performed on the analog device via natural time evolution of, say, the voltage on an electrical capacitor. This addresses the computational bottleneck of numerical integration of dynamics with digital solvers since the analog system naturally performs this integration.

The analog system is composed of repeated subunits, or unit cells, where the number of unit cells is equal to the dimensionality of the problem that the analog system is configured to solve. Each unit cell is composed of a thermal noise source and resistive and capacitive circuit elements. Moreover, each unit cell can, in principle, be coupled (via a capacitive bridge) to every other unit cell. An arbitrary connectivity matrix (analog to the adjacency matrix in graph theory) describes how the unit cells are coupled to each other. This allows the connectivity to be tailored to the geometry of the problem at hand.

As noted above, inductive bias reduces training data requirements and training complexity. Our system enables the user to incorporate inductive bias into the model. In particular, the connectivity matrix should be closely connected to the geometry of the problem to maintain a strong inductive bias. Indeed, in our system, simple switches in the connection bridges allow connections to be turned on or

off, allowing the user to upload the geometry of the problem onto the analog device.

Our system can be operated in pre-trained mode, where the score network is already trained, or in training mode, where the score network is first trained using training data produced by our analog device. In our system, a digital device uploads the data onto the analog device by appropriately charging the capacitors of each unit cell. These capacitor charges then become the dynamical variables of interest, evolving under the corresponding forward or reverse process.

One aspect of our system is rooted in the thermodynamic concept of Maxwell's Demon. The second law of thermodynamics says that global entropy does not decrease over time, but a Maxwell's Demon can locally reduce the entropy of a system by making observations on that system and adaptively interacting with the system. An electrical version of Maxwell's Demon can reduce the entropy of the set of unit cells in our analog device, specifically by observing the capacitors' charges in the unit cells and adjusting an applied voltage in the cells. As a result, our system can physically implement the dynamics in the Reverse SDE, which tend to reduce entropy and thus at first look contradictory with the second law of thermodynamics. Our Maxwell's Demon can be implemented as a digital device, such as a central processing unit (CPU) or a field-programmable gate array (FPGA), that stores the trained score network and interacts continuously with the analog device. In this sense, the Maxwell's Demon (or digital score network) acts as an AI agent who intelligently interacts with a thermodynamic physical system. Taken together, the analog system and the digital AI agent/Maxwell's Demon form a thermodynamic AI system for generative modeling.

5 The Unit Cell

5.1 Warm-up exercise: The RC circuit

Before introducing our unit cell, we first consider a simple RC circuit, i.e., a circuit with a resistor R and a capacitor C in series. The current law states that the current in the resistor branch, I_R, should be equal in magnitude and of opposite sign to the current in the capacitor branch, I_C. Thus, we have $I_R = -I_C$. The voltage law states that the sum of the voltages, v_R and v_C, across both branches should be zero, thus $v_R = -v_C$. Noting that

$$I_R = \frac{v_R}{R} \quad \text{and} \quad I_C = C\frac{\mathrm{d}v_C}{\mathrm{d}t}, \tag{14}$$

we get the following:

$$-v_C/R = C\frac{\mathrm{d}v_C}{\mathrm{d}t}. \tag{15}$$

Rearranging to the differential form gives:

$$-C\mathrm{d}v_C = \frac{v_C \mathrm{d}t}{R}. \tag{16}$$

This provides a simple building block for first-order differential equations to be solved analogically.

5.2 Unit cell involving voltage mixers and fixed resistors

Constructing circuits for SDEs involves a source of Brownian noise. This can be provided by a second resistor R', which we suppose has much higher noise than R and therefore emits voltage noise v_w while R does not emit any voltage noise. Hence, we consider a circuit with capacitor C, resistor R, and resistor

R' all in series. For this circuit, we obtain the differential form:

$$-C\mathrm{d}v_C = \frac{v_C\mathrm{d}t + \mathrm{d}w}{R + R'} \tag{17}$$

which corresponds to a SDE of order 1, where we have used $v_w\mathrm{d}t = \mathrm{d}w$. We identify the drift term $f(v) = v_C/(R + R')$ and the diffusion term $g = 1/(R + R')$.

However, these terms are not time-dependent, hence they may not be general enough to simulate SDEs of the form of the score SDE (as discussed in Sec. 2.1).

Fig. 4 shows the RC circuit with two voltage mixers, which add in time-dependence of the drift and diffusion terms. The two voltage mixers multiply the circuit voltages before and after resistor R by time-dependent voltages $k(t)$ and $h(t)$, respectively.

Next, consider a complete analysis of the circuit in Fig. 4. For this analysis, we assume w is an arbitrary voltage noise source with noise voltage v_w, the voltage sources $h(t)$ and $k(t)$ are time-dependent voltage sources with voltages v_k and v_h. We assume that the analog mixers are ideal multiplicative mixers, that is, their output voltage $v_{\text{out}}(v_a, v_b)$ is equal to the product of the two input voltages v_a and v_b, as $v_{\text{out}}(v_a, v_b) = v_a v_b$.

The voltage law for the main loop reads:

$$v_w + v_{M_{12}} + v_R + v_{M_{34}} + v_C = 0 \tag{18}$$

The ideal mixer voltages are defined as follows:

$$v_{M_{12}} = v_2 - v_1 = v_1 v_5 - v_1, \tag{19}$$

$$v_{M_{34}} = v_4 - v_3 = v_3 v_6 - v_3, \tag{20}$$

where v_i refers to the voltage on circuit node i. These node voltages can be associated with the difference in voltage across components as follows:

$$\begin{aligned} v_1 &= v_w, \\ v_5 &= v_k, \\ v_6 &= v_h, \\ v_3 &= v_w + v_{M_{12}} + v_R. \end{aligned}$$

The mixer voltages become:

$$v_{M_{12}} = v_w(v_k - 1), \tag{21}$$

$$v_{M_{34}} = (v_R + v_w v_k)(v_h - 1). \tag{22}$$

The voltage law now reads:

$$v_C + v_R v_h + v_w v_h v_k = 0. \tag{23}$$

Noting that, using Ohm's law, the current-voltage relations for the resistor and capacitor are

$$I_R = \frac{v_R}{R} \text{ and } I_C = C\frac{\mathrm{d}v_c}{\mathrm{d}t} \tag{24}$$

and that the current law in the circuit states that

$$I_R = I_C - I_h, \tag{25}$$

we get the following relation

$$\frac{v_R}{R} = C\frac{\mathrm{d}v_c}{\mathrm{d}t} - I_h \tag{26}$$

$$v_R = RC\frac{\mathrm{d}v_c}{\mathrm{d}t} - RI_h. \tag{27}$$

Replacing this in equation (23) gives the following:

$$v_C + v_h(RC\frac{\mathrm{d}v_c}{\mathrm{d}t} - RI_h) + v_w v_h v_k = 0. \tag{28}$$

The current through the voltage source v_h, I_h, is

$$I_h = \frac{v_h}{R_{M_6}}, \tag{29}$$

where R_{M_6} is the input resistance to the mixer from node 6. In the case where the input resistance of the mixer is very large (often the case in practice), the current going into the mixer from node 6 is very small and can be neglected. After rearranging the terms in the differential form, we get

$$-RC\mathrm{d}v_C = \frac{v_C\mathrm{d}t}{h(t)} + k(t)\mathrm{d}w, \tag{30}$$

where we have used $v_k = k(t), v_h = h(t)$ and $v_w\mathrm{d}t = \mathrm{d}w$. We therefore have tunable drift and diffusion terms, with $h(t)$ controlling the drift term and $k(t)$ controlling the diffusion term. These tunable drift and diffusion terms are useful for implementing the forward and reverse SDEs for diffusion models on hardware.

The probabilistic flow ODE can also be implemented in an RC circuit corresponding to Eq. (16) by adding a voltage mixer to the voltage at a point between the resistor and the capacitor. Adding this voltage mixer yields the following ODE:

$$-RC\mathrm{d}v = \frac{v}{h(t)}\mathrm{d}t. \tag{31}$$

By adding in a second voltage equal to $1/2G(t)^2\mathbf{s}_{\boldsymbol{\theta}}$ (related to the second term in Eq. (12)), it is possible to simulate the behavior of the probabilistic flow ODE given in Eq. (13) to generate new samples. This is explained in more detail in section 11.2 for N unit cells.

5.3 Unit cell involving time-varying resistors

An alternative version of our unit cell does not use voltage mixers. In practice, voltage mixers involve multiple components, and hence avoiding voltage mixers reduces circuit complexity.

Instead of using voltage mixers, time-varying resistors can be used to introduce time dependence into the drift and diffusion terms. One way of constructing a time-varying resistor is with a field effect transistor (FET) or with a network of FETs.

With a FET, we can use a voltage applied to the gate of the FET to control the resistance of the FET. By applying a time-varying voltage to the gate of the FET, within its linear operation range, we then get a time-varying resistor.

A voltage-controlled resistor, such as a FET, can also be used to manipulate the gain of a simple non-inverting amplifier. This is possible because the voltage gain of this amplifier is determined by a resistive voltage divider, and so if one of the resistances is time varying, the gain becomes time-varying.

Consider the circuit in Fig. 8. In this circuit, a first FET acts as a voltage-controlled resistor in the feedback network of a non-inverting amplifier to control the noise source (diffusion term). A second FET in parallel with the capacitor controls the drift term.

The voltage law analysis gives:

$$v_w + v_R + v_{\rho_1} = 0 \tag{32}$$

and

$$v_{\rho_1} = v_C + v_{R_C}. \tag{33}$$

The current law analysis gives

$$I_R = I_{\rho_1} + I_C. \tag{34}$$

Using equations (32) and (33) with (34) and the current-voltage relations of the resistor and capacitor, we get

$$\begin{aligned}
\frac{v_w}{R} &= \frac{v_{\rho_1}}{\rho_1(t)} + C\dot{v}_c && (35)\\
&= \frac{v_C + V_{R_C}}{\rho_1(t)} + C\dot{v}_c && (36)\\
&= \frac{v_C + R_C I_C}{\rho_1(t)} + C\dot{v}_c && (37)\\
&= \frac{v_C + R_C C\dot{v}_C}{\rho_1(t)} + C\dot{v}_c. && (38)
\end{aligned}$$

Finally, the noise voltage after amplification is

$$v_w = w\left[1 + \frac{R^*}{\rho_2(t)}\right]. \tag{39}$$

Using this relation and rearranging, we finally get

$$-C\frac{\mathrm{d}v_C}{\mathrm{d}t}\left[1 + \frac{R_C}{\rho_1(t)}\right] = \frac{v_C}{\rho_1(t)} + \frac{w}{R}\left[1 + \frac{R^*}{\rho_2(t)}\right]. \tag{40}$$

In the above, we used $\dot{v}_C = \mathrm{d}v_C/\mathrm{d}t$. We can rearrange and get the following SDE:

$$-C\mathrm{d}v_C\left[1 + \frac{R_C}{\rho_1(t)}\right] = \frac{v_C}{\rho_1(t)}\mathrm{d}t + \frac{\mathrm{d}w}{R}\left[1 + \frac{R^*}{\rho_2(t)}\right], \tag{41}$$

where we have independently addressable drift and diffusion terms by tuning the variable resistances $\rho_1(t)$ and $\rho_2(t)$. Note that this result is analogous to the SDE in equation (130), albeit with slightly different coefficients.

5.4 Alternative noise sources

As mentioned in the previous sections, implementing SDEs in electronics involves a source of Gaussian noise in voltage or current. In general, the amplitude of this noise should be controllable independent of

the other circuit parameters. The circuits outlined in this document have used an arbitrary noise source. In practice, this arbitrary noise source can be of various types.

Specifically, note that the circuit diagrams in Figure 4 and Figure 5 include an abstract noise source denoted w. Physically speaking, a physical circuit element can represent the abstract noise source given by w. A thermal noise source or a shot noise source are possible physical realizations of w.

One type of possible noise is thermal noise. Thermal noise, also called Johnson-Nyquist noise, comes from the random thermal agitation of the charge carriers in a conductor, resulting in fluctuations in voltage or current inside the conductor. Thermal noise is Gaussian and has a flat frequency spectrum (white noise) with fluctuations in the voltage of standard deviation

$$v_{tn} = \sqrt{4k_B T R \Delta f}, \tag{42}$$

where R is the resistance of the conductor, k_B is the Boltzmann constant, T is the absolute temperature and Δf is the frequency bandwidth. The amplitude of the voltage fluctuations can be controlled by changing the temperature or the resistance. In practice, a thermal noise source can be implemented using a large resistor in series with a voltage amplifier. Alternatively, a thermal noise source can be implemented using a tunable resistor (transistor operated in the linear regime) where the noise amplitude scales with the value of the resistance.

Another type of noise is shot noise. Shot noise arises from the discrete nature of charge carriers and from the fact that the probability of a charge carrier crossing a point in a conductor at any time is random. This effect is particularly notable in semiconductor junctions where the charge carriers should overcome a potential barrier to conduct a current. The probability of a charge carrier passing over the potential barrier is an independent random event. This induces fluctuations in the current through the junction. Shot noise is Gaussian and has a flat frequency spectrum (white noise) with fluctuations in the current of standard deviation

$$I_{sn} = \sqrt{2q|I|\Delta f}, \tag{43}$$

where I is the current through the junction, $\mathbf{q}$ is the electron charge and Δf is the frequency bandwidth. The amplitude of the current fluctuations can be controlled by changing the magnitude of the DC current passing through the junction. In practice, a source of shot noise would be implemented using a pn diode (for example, a Zener diode in reverse bias configuration) in series with a controllable current source.

6 Two Coupled Unit Cells

As mentioned above, incorporating the problem geometry into the noise process contributes to reducing the strain on the score network, as this can be viewed as providing an inductive bias for the model. In order to incorporate geometric information (or geometric correlations) into the analog circuitry, there should be correlations between cells in the drift and diffusion terms. We achieve these correlations by coupling unit cells capacitively.

Consider the circuit in Fig. 6, where two unit cells are connected through the coupling capacitor C_{12}. In the following derivation, we analyze the case where the unit cells are composed of multiplicative mixers. However the capacitive coupling formalism extends to two coupled unit cells with variable resistors. Fig. 7 shows the circuit diagram associated with two coupled cells in the variable resistor case.

Analyzing the circuit in Fig. 6 using the method of section 5.2 yields the following set of coupled

SDEs:

$$-\frac{\mathrm{d}v_{C1}}{\mathrm{d}t}(C_2C_{12}+C_1C_{12}+C_1C_2)=\frac{C_2+C_{12}}{R_1}\left(k_1w_1+\frac{v_{C1}}{h_1}\right)+\frac{C_{12}}{R_2}\left(k_2w_2+\frac{v_{C2}}{h_2}\right) \tag{44}$$

$$-\frac{\mathrm{d}v_{C2}}{\mathrm{d}t}(C_2C_{12}+C_1C_{12}+C_1C_2)=\frac{C_2+C_{12}}{R_2}\left(k_2w_2+\frac{v_{C2}}{h_2}\right)+\frac{C_{12}}{R_1}\left(k_1w_1+\frac{v_{C1}}{h_1}\right). \tag{45}$$

To simplify the notation, these equations can be written in matrix form as follows

$$-\dot{\mathbf{v}}(t)=\mathbf{C}^{-1}\mathbf{R}^{-1}\left[\mathbf{h}(t)^{-1}\mathbf{v}(t)+\mathbf{k}(t)\mathbf{w}(t)\right], \tag{46}$$

where

$$\dot{\mathbf{v}}(t)\equiv\begin{bmatrix}\frac{\mathrm{d}v_{C1}}{\mathrm{d}t}\\ \frac{\mathrm{d}v_{C2}}{\mathrm{d}t}\end{bmatrix},\ \mathbf{C}\equiv\begin{bmatrix}C_1+C_{12} & -C_{12}\\ -C_{12} & C_2+C_{12}\end{bmatrix},\ \mathbf{R}\equiv\begin{bmatrix}R_1 & 0\\ 0 & R_2\end{bmatrix},\ \mathbf{w}(t)\equiv\begin{bmatrix}w_1\\ w_2\end{bmatrix},$$
$$\mathbf{h}(t)\equiv\begin{bmatrix}h_1 & 0\\ 0 & h_2\end{bmatrix},\ \mathbf{k}(t)\equiv\begin{bmatrix}k_1 & 0\\ 0 & k_2\end{bmatrix},\ \text{and}\ \mathbf{v}(t)\equiv\begin{bmatrix}v_{C1}\\ v_{C2}\end{bmatrix}.$$

Due to the non-zero off-diagonal elements of the capacitance matrix, $\mathbf{C}$, we get a coupled system of equations with correlations in both the drift and diffusion terms. With each cell having its own control voltages, $h_i(t)$ and $k_i(t)$ for $i\in\{1,2\}$, each cell's drift and diffusion terms remain independently addressable with the proper choice of $\mathbf{h}(t)$ and $\mathbf{k}(t)$.

A simplified version of our circuit has been used (see Fig. 8) to study the voltage correlations between two capacitively coupled RC circuits. There are strong correlations in the random walks of the capacitors' voltages in the two circuits, which suggest that our (somewhat more complicated) circuits should lead to strong correlations between unit cells.

7 Problem Geometry and the Connectivity Matrix

As illustrated in Fig. 9, different problems have different geometries. Time series data and DNA sequence data are both one-dimensional (1D); there is a 1D notion of locality for these datasets. Images are 2D, and solutions to partial differential equations in real space are often 3D. Other problems are best described by graphs, rather than grids, such as molecular structure problems.

Ideally, we would like to account for the problem geometry in the noise process, i.e., in the drift and diffusion terms appearing in the differential equations. This would ease the burden on the score network, so that the score network is not forced to generate all of the correlations in the data. Ultimately this could translate into faster training and less training data for the score network.

Now let us consider problem geometry in the context of an array of unit cells, where each unit cell is constructed as in section 5. Namely, consider N RC cells, where pairs of such cells may be capacitively coupled as in section 6.

We can represent the connectivity of such a circuit through a combinatorial graph $G(V,E)$. We represent the connectivity between vertices V with the adjacency matrix $\mathcal{A}$ of $G(V,E)$. If $\mathcal{A}_{ij}=1$, then cells i and j are coupled, while $\mathcal{A}_{ij}=0$ when the pair is uncoupled.

We achieve such connectivity in the physical circuit by including switches on each coupling wire (e.g., the wire containing capacitor C_{12} in FIG. 8 and as shown in FIG. 10, described below). These switches could be implemented using transistors in a physical device, and hence could be voltage controlled.

These switches can be open or closed according to the adjacency matrix $\mathcal{A}$. Given an adjacency

matrix $\mathcal{A}$, the 2×2 capacitance matrix defined in the 2-cell case generalizes to the $N \times N$ matrix:

$$\mathbf{C}_{ij} = \begin{cases} C_i + \sum_{\{k:\mathcal{A}_{ik}=1\}} C_{ik} & \text{if } i = j \\ -C_{ij} & \text{if } i \neq j \end{cases} \tag{47}$$

with C_i the capacitance of the capacitor in cell i, and C_{ij} the capacitance of the branch coupling i and j. In the event that every cell is pairwise coupled to all other cells, the matrix $\mathbf{C}$ is a fully dense symmetric matrix. Arbitrary symmetric matrices can be constructed by specifying an arbitrary adjacency matrix $\mathcal{A}$.

In many applications, the circuit geometry is specified by a sparse adjacency matrix. For example, if N cells are arranged on an d-dimensional lattice, each cell is coupled only to its nearest neighbors, and there are $2d$ nearest neighbors for each unit cell. As d is typically much less than N, then the adjacency matrix will be sparse. For example, the matrix will be tridiagonal if $d = 1$.

8 Differential Equations for the Forward Process

Combining ideas from the previous sections makes it possible to build a circuit of N unit cells, coupled according to an arbitrary adjacency matrix $\mathbf{A}$, which allows physical simulation of the SDEs that are commonly used in generative modeling. In this section, we present how such an analog system can be used to simulate the forward process of a generative diffusion model.

The first step to simulate the forward SDE is to upload a data point onto the analog device. This involves encoding a data point $\mathbf{x} = \{x_1, x_2, ..., x_N\}$, which is composed of N feature values, into an N-dimensional voltage vector $\mathbf{v}(t) = \{v_1(t), v_2(t), ..., v_N(t)\}$, where the entry $v_i(t)$ is the voltage across the capacitor in the ith unit cell at time t. Specifically, for the forward process, the data point at the initial time $t = 0$ are uploaded so that there is a direct mapping $x_i \to v_i(0)$ for each i. The nature of this mapping from the data feature value to a voltage depends on the dataset; for images, the values of the pixels can be converted to voltages in a certain voltage window. The forward process evolves the vector $\mathbf{v}(t)$ over time, and eventually at a later time t this vector is downloaded and converted back to a vector of feature values.

Next, we analyze the case where the unit cell is constructed with voltage mixers and fixed resistors. However, the treatment in this section extends straightforwardly to other variants of the unit cell, such as the unit cell based on variable resistors.

Each unit cell provides one Brownian source voltage in series with its resistor, and the N-tuple of all such signals is an N-dimensional Brownian process $\mathbf{w}(t)$. The following construction generalizes the two-cell system: the $N \times N$ matrix $\mathbf{C}$ defined in Eq. (47) is the general Maxwell capacitance matrix of the coupled circuit, and the resistances of each cell are collected in the N-vectors $\mathbf{R} \equiv \text{diag}(R_1(t), R_2(t)..., R_N(t))$ and $\mathbf{R}' \equiv \text{diag}(R'_1(t), R'_2(t)..., R'_N(t))$. Each individual cell j is fed time-dependent voltage sources $h_j(t)$ and $k_j(t)$ at the nodes labeled in Fig. 4. As in the case of two coupled cells, we collect these signals along the diagonal of $N \times N$ matrices $\mathbf{k}(t) \equiv \text{diag}(k_1(t), k_2(t)..., k_N(t))$ and $\mathbf{h}(t) \equiv \text{diag}(h_1(t), h_2(t)..., h_N(t))$ With this circuit architecture, the vector $\mathbf{v}(t) \equiv (v_1(t), v_2(t)..., v_N(t))$ collecting the voltage across each cell's capacitor at a time $t > 0$ is an N-dimensional random variable. This random variable evolves according to the SDE:

$$-\mathrm{d}\mathbf{v}(t) = \mathbf{C}^{-1}\mathbf{R}^{-1}\left[\mathbf{h}(t)^{-1}\mathbf{v}(t)\mathrm{d}t + \mathbf{k}(t)\mathrm{d}\mathbf{w}\right], \tag{48}$$

If $\mathbf{v}(0)$ is the vector of voltage measurements across each cell's capacitor an initial time t_0, $\mathbf{v}(t)$ is the

solution to the following integral equation for any $t > 0$;

$$\mathbf{v}(t) = \mathbf{v}(0) - \int_0^t \left[\mathbf{C}^{-1}\mathbf{R}^{-1}\mathbf{h}(s)^{-1}\mathbf{v}(s)\right]\mathrm{d}s + \int_0^t \left[\mathbf{C}^{-1}\mathbf{R}^{-1}\mathbf{k}(s)\right]\mathrm{d}\mathbf{w} \tag{49}$$

The second integral is treated as an Ito integral. The signal matrices $\mathbf{h}(t)$ and $\mathbf{k}(t)$ appear only in the drift and diffusion functions, respectively. Hence, by controlling such source signals, we can simulate any linear differential of the form (we omit the time-dependencies of $\mathbf{v}$ and $\mathbf{w}$ for readability):

$$\mathrm{d}\mathbf{v} = \mathbf{L}(t)\mathbf{v}\mathrm{d}t + \mathbf{M}(t)\mathrm{d}\mathbf{w} \tag{50}$$

where $\mathbf{L}(t)$ and $\mathbf{M}(t)$ are $N \times N$ matrices whose entries do not depend on the value of $\mathbf{v}$ and are given by:

$$\mathbf{L}(t) = -\mathbf{C}^{-1}\mathbf{R}^{-1}\mathbf{h}(t)^{-1} \tag{51}$$

$$\mathbf{M}(t) = \mathbf{C}^{-1}\mathbf{R}^{-1}\mathbf{k}(t)\,. \tag{52}$$

We dropped the minus sign from the diffusion term in Eqs. (48) and (50) since $\mathrm{d}\mathbf{w}$ is Gaussian distributed with zero mean, and hence the minus sign would have no effect on the term.

We therefore obtain a system that can simulate any forward SDE of the form of Eq. (1), provided the diffusion matrix $\mathbf{G}$ in Eq. (1) has no dependence on the voltage, and the drift term $\mathbf{f}$ in Eq. (1) depends linearly on the voltage. This therefore includes state-of-the-art diffusion models, such as those represented by Eq. (7) where $\mathbf{G}(t) \to g(t)$ is a scalar and the drift term is linear. It also includes diffusion models with inductive biases, since correlations can be introduced by the $\mathbf{C}$ matrix that depends on the connectivity of the full circuit.

The primary role for the forward process in diffusion models is to generate sample trajectories, and these trajectories serve as training data used to evaluate the loss function associated with training the score network. Hence, the intention of introducing an analog device for the forward process, described by Eq. (50), is to efficiently generate training data to allow for efficient evaluation of the loss function, such as the loss functions in Eq. (8) or Eq. (10).

8.1 Example: Denoising Diffusion Probabilistic Models

To see how to use our circuit architecture to perform generative modeling, consider the forward SDE used in DDPM (Eq. (3)). This model introduces the matrix parameter

$$\mathbf{B}(t) = \mathrm{diag}(\beta_1(t), \beta_2(t)..., \beta_N(t)) \tag{53}$$

and its square root

$$\hat{\mathbf{B}}(t) = \sqrt{\mathbf{B}(t)} = \mathrm{diag}(\sqrt{\beta_1(t)}, \sqrt{\beta_2(t)}..., \sqrt{\beta_N(t)}) \tag{54}$$

as a linear drift operator and diffusion matrix, respectively. Our circuit is equipped to simulate such linear drift/diffusion equations, and choices of $\mathbf{L}(t) = -\frac{1}{2}\mathbf{B}(t)$ and $\mathbf{M}(t) = \hat{\mathbf{B}}(t)$ in Eqs. (51) and (52) realize the DDPM forward process. As neither the drift linear operator nor diffusion matrix have off-diagonal values, a completely uncoupled connectivity structure is sufficient to program the correct form of the forward process. In particular, given diagonal matrices $\mathbf{C}, \mathbf{R}$ determined completely by the properties of the electrical parts, the time-dependent voltages (used in the unit cells associated with the voltage

mixer approach) should have the following form:

$$h_i(t) = \frac{2}{C_i R_i \beta_i(t)} \quad \text{and} \quad k_i(t) = C_i R_i \sqrt{\beta_i(t)}. \tag{55}$$

9 Noisy Distribution (Stationary Distribution of the Forward Process)

Consider the forward process with the SDE given by Eq. (1). Suppose that the coefficients $f(\mathbf{x}, t)$ and $\mathbf{G}(t)$ are such that the process is ergodic (i.e., for any points x_1 and x_2 and any time interval $[0, T]$, there is a non-zero probability that a sample trajectory of the process based at x_1 at $t = 0$ reaches x_2 at $t = T$). Under this ergodic assumption, the forward process possesses a unique stationary distribution $p_{noise}(\mathbf{x})$; any initial distribution $p_{data}(\mathbf{x})$ of the forward process converges towards p_{noise} as $t \to \infty$. In practice, we may only have access to training samples from the initial distribution $p_{data}(\mathbf{x})$, but we should have access to the functional form of the stationary distribution $p_{noise}(x)$. Our system perturbs the training samples under a given drift/diffusion model, and the distribution of the perturbed training samples begins to look more and more like p_{noise} as the forward process continues longer. The reverse process is the process used to carry samples from the distribution p_{noise} to samples approximately distributed according to p_{data}. This is useful, for we can now sample from the unknown $p_{data}(\mathbf{x})$ that the training data comes from by sampling from a known distribution p_{noise}; we know where any choice of unknown p_{data} will end up after the forward process, and we learn this distribution by retracing how it got there.

10 The Score Network as a Maxwell's Demon

The second law of thermodynamics states that the entropy of an isolated system cannot decrease over time. At first sight, this might look like an obstacle to implementing the reverse process in diffusion models in a physical system. After all, the reverse process tends to reduce the entropy of the data, by turning highly noisy data into highly structured data. However, this simply means that we cannot implement the reverse process with *an isolated physical system.*

It is possible to reduce entropy locally by increasing entropy somewhere else. For example, this is the basis for refrigerators. This is also the basis for a thought experiment called Maxwell's Demon (MD). In the MD experiment, an intelligent observer can spatially separate the two components of a gas mixture, going from a high entropy situation to a low entropy situation, as shown in the left panel of Fig. 11. To accomplish this, the observer watches the gas particles to see which component is approaching the barrier and then removes the barrier when it would help the sorting process.

Our circuit for solving the reverse process in diffusion models is a physical example of the Maxwell's Demon scenario. Instead of the positions of gas particles, the dynamical variables are the voltages across the capacitors in each of the unit cells. Even if these voltages start in a high entropy state (e.g., being uncorrelated), they can evolve over time towards a low entropy state (e.g., being strongly correlated).

This entropy reduction is facilitated by a neural network, called the score network. The score network can take various forms. For example it could be stored on and executed by a digital Field Programmable Gate Array (FPGA), or it could be stored in a memory and executed by a digital CPU operably coupled to the memory. The score network could even be an analog device, as discussed below in Sec. 15.

This neural network / processing device combination acts as the demon in the Maxwell's Demon scenario in that it continuously observes the state of the analog system and adapts the applied voltage in each circuit appropriately. This is illustrated in the right panel of Fig. 11. It therefore makes sense to refer to our system as a thermodynamic AI system. After all, our system exploits an artificial intelligence unit

(the neural network on the CPU or FPGA) in order to locally reduce the entropy, in a thermodynamic sense, of the analog system.

Fig. 12 shows details of how the score network interacts with the analog system. As shown in Fig. 12, a voltmeter reads off the voltage across the capacitor in each unit cell, and the set of voltages forms the state vector, denoted x. Each of these voltages is converted into a digital signal via an analog-to-digital converter (ADC) and these digital signals are sent to the CPU or FPGA. Inside the CPU or FPGA, the score network is evaluated at the point associated with the inputted signal. The output of the score network is the predicted score, which is a vector (the score vector). This score vector is converted from a digital signal into an analog signal via a digital-to-analog converter (DAC), and then it becomes a vector of voltages. Each element of this voltage vector corresponds to a voltage that is applied inside of the corresponding unit cell. This entire process happens continuously, in real time, during the time evolution associated with the reverse process of the diffusion model.

Figure 13A and 13B delve deeper into how the output of the score network gets incorporated into each unit cell as an applied voltage. The main issue here is how to multiply the output of the score network by the function $g(t)^2$ that appears in the Reverse SDE or Reverse ODE. (More generally, the function $g(t)^2$ is replaced by $\mathbf{G}(t)\mathbf{G}(t)^\intercal$ when $\mathbf{G}$ is a matrix.)

Figure 13A and 13B depict two alternative methods for coupling the score network to the analog unit cell, for solving the Reverse process. One method, shown in Fig. 13A, is analog in spirit. Here, the multiplication by $g(t)^2$ is done physically in the circuit with analog components. The other method, shown in Fig. 13B, is digital in spirit. Here, the multiplication by $g(t)^2$ is done digitally on the CPU or FPGA, and then the resulting digital signal is fed into the ADC and directly applied as a voltage in the unit cell. There are pros and cons to both of these methods. The analog method is more elegant as it directly uses physical systems for the multiplication, while the digital method uses fewer analog components and hence can be more convenient.

11 Equations for Reverse Process

11.1 SDE Equations for Reverse Process

Suppose that the SDE associated with the forward process is given by Eq. (50), as discussed in Section 8. In this case, the corresponding SDE for the reverse process is given by:

$$d\mathbf{v}(t) = \Big(\mathbf{L}(t)\mathbf{v} - \mathbf{M}(t)\mathbf{M}(t)^\intercal \nabla \log p(\mathbf{v}, t)\Big)dt + \mathbf{M}(t)\mathrm{d}\overline{\mathbf{w}}_t\,. \tag{56}$$

Here, t indexes time but moves in reverse, beginning from T and incrementing negatively to 0, with each dt being a negative increment. Also, $\overline{\mathbf{w}}_t$ is a reverse Brownian motion indexed by t.

At $t = T$, we initialize a pass of reverse model by sampling $v(T) \sim p_{noise}$, and begin a physical clock recording time τ elapsed since the beginning of the reverse process. τ is the physical time variable incrementing postively from 0 to T as the reverse process is performed, while t is an abstract time moving against the physical flow of time.

We make the substitution $t \to T - \tau$. This way, $\mathbf{v}(T - \tau)$ is the voltage value τ seconds into the reverse process started at $\mathbf{v}(T)$. The coefficients of the the drift and diffusion terms are also evaluated at $T - \tau$, corresponding to τ seconds into the reverse process. Also, dt is replaced by $d(T - \tau) = -d\tau$, and $d\overline{\mathbf{w}}_t$ is replaced by the forward Brownian motion $d\mathbf{w}_\tau$. The SDE now reads:

$$d\mathbf{v}(T-\tau) = -\Big(\mathbf{L}(T-\tau)\mathbf{v}(T-\tau) - \mathbf{M}(T-\tau)\mathbf{M}(T-\tau)^\intercal \nabla \log p(\mathbf{v}, T-\tau)\Big)d\tau + \mathbf{M}(T-\tau)d\mathbf{w}_\tau. \tag{57}$$

Ideally we would like to solve this equation, but in practice one can approximate the score function with the score network $s_\theta(\mathbf{v}, T-\tau)$. This approximation leads to the following SDE:

$$d\mathbf{v} = \Big(-\mathbf{L}(T-\tau)\mathbf{v} + \mathbf{e}(\mathbf{v}, T-\tau)\Big)d\tau + \mathbf{M}(T-\tau)\mathrm{d}\mathbf{w}_\tau \tag{58}$$

where we write $\mathbf{v}$ instead of $\mathbf{v}(T-\tau)$ for simplicity, and also define

$$\mathbf{e}(\mathbf{v}, T-\tau) \equiv \mathbf{M}(T-\tau)\mathbf{M}(T-\tau)^\intercal \mathbf{s}_\theta(\mathbf{v}, T-\tau)\,. \tag{59}$$

The reverse process can be simulated by integrating the SDE given in Eq. (58) in our electrical hardware. We have already shown how to integrate the linear SDE (30) on analog electrical hardware, and after evaluating all drift/diffusion terms at decreasing time values $T-\tau$ rather than the physical time τ, the SDE of the reverse process differs from its forward counterpart only by a non-linear drift term given by the $\mathbf{e}(\mathbf{v}, T-\tau)$ function.

As discussed above in Section 10, Figures 13A and 13B provide two alternative methods to program the approximated non-linear drift piece $\mathbf{e}(\mathbf{v}, T-\tau)$ into the pre-existing circuit model simulating the forward SDE. Both of these methods involve digital evaluations of $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}, T-\tau)$ passed into the circuit unit cells as a voltage signal, via DACs. However, the two methods differ in how the multiplicative factor $\mathbf{M}(T-\tau)\mathbf{M}(T-\tau)^\intercal$ is incorporated into the $\mathbf{e}(\mathbf{v}, T-\tau)$ function. Figure 13B shows the case where this multiplication is done digitally, where $\mathbf{M}(T-\tau)\mathbf{M}(T-\tau)^\intercal$ is multiplied times $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}, T-\tau)$ on the digital computer, and the resulting signal is passed through a DAC and then applied as a voltage to the circuit. Alternatively, Figure 13A shows an example of how to do this multiplication with an analog circuit, in the special case where $\mathbf{M}(T-\tau)$ is diagonal and hence can be thought of as a set of scalar quantities (each of the form $g(T-\tau)$). For each i, we port $\mathbf{e}(\mathbf{v}, T-\tau)_i$ to the ith unit cell.

Hence, we encode the integral equation in the stochastic dynamics of the circuit. Let us now consider the integral equation. Suppose the score network $\mathbf{s}_\theta(\mathbf{v}, T-\tau)$ is trained on data from a forward process that is stopped after time T. If the reverse process is initialized at voltages $\mathbf{v}(T) \sim p_{noise}$ at $t = T$, the voltage at a time τ seconds into the reverse process, $\mathbf{v}(T-\tau)$ is given by the integral equation:

$$\mathbf{v}(T-\tau) = \mathbf{v}(T) + \int_0^\tau \Big(\mathbf{L}(T-\tau')\mathbf{v} + \mathbf{e}(\mathbf{v}, T-\tau')\Big)d\tau' + \int_0^\tau \mathbf{M}(T-\tau')\mathrm{d}\mathbf{w}_{\tau'}, \tag{60}$$

If we run the reverse process up to time $\tau = T$, the solution to the integral equation at this time is a voltage vector $\mathbf{v}(0)$ approximately distributed by p_{data}. The final value $\mathbf{v}(0)$ of the reverse process is not exactly distributed according to p_{data}, but the approximation becomes more precise as $T \to \infty$ and as the score network is trained more accurately. Hence, the reverse process perturbs samples from an initial distribution $\mathbf{v}(T) \sim p_{noise}$ back towards samples that are approximately distributed according to the unknown p_{data}.

11.2 Probabilistic flow ODE for N cells

As referred to in sections 2.5 and 5, the probabilistic flow ODE can be used for sample generation. For the system of N cells presented above and an added voltage signal $\mathbf{e}(\mathbf{v}, t) = -(1/2)\mathbf{G}(t)\mathbf{G}(t)^\intercal \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}, t)$, the probabilistic flow ODE reads

$$d\mathbf{v}(t) = \Big(\mathbf{M}(t)\mathbf{v} + \mathbf{e}(\mathbf{v}, t)\Big)dt. \tag{61}$$

New data samples $\mathbf{x}$ can therefore be generated in exactly the same way than when using the reverse SDE. The performance of this approach will depend on the noise level of the system, as it supposes that there is no noise stemming from the RC circuit.

12 Switching between the forward and reverse processes

Ideally, the same physical device (e.g., the same electrical chip) can be used for the forward and reverse processes, as opposed to having two separate physical devices.

However, comparing the unit cell circuits in Fig. 4 and Fig. 13, which pertain to the forward and reverse processes, respectively, shows that the unit cells are not exactly the same. In particular, the unit cell for the reverse process has additional circuit elements that are not used in the forward process.

The circuit in Fig. 28 addresses this issue. Fig. 28 shows the unit cell in Fig. 13A, but with switches added in various locations. These switches can make it possible to toggle a unit cell between the forward process and the reverse process. Namely, when the switches are open, the unit cell corresponds to the one for forward process, and when the switches are closed it corresponds to the one for the reverse process. In practice, the switches can be made from voltage-controlled transistors.

Hence, by adding in these switches, one can use the same physical device for both the forward and reverse processes.

Fig. 28 illustrates this concept for a unit cell construction based on voltage mixers. However, this concept applies more broadly to other unit cell constructions (e.g., the construction based on variable resistors). Indeed, switches can be added to other other unit cell constructions for toggling between the forward and reverse processes.

13 Flowchart of the entire system (with digital score network)

Next we present our entire thermodynamic AI system for generative modeling. We will discuss two scenarios in which we will apply our system: (1) A scenario in which the score network is already trained, (2) A scenario in which the score network is untrained.

Figure 14 shows a flowchart for the scenario is which the score network has already been trained. In this case, we do not need to run the forward process. We simply run the reverse process to generate new datapoints. The digital device stores prior information in the form of a connectivity matrix (1402). This information is uploaded to the analog device by opening or closing the relevant switches (1406), e.g., as shown in Fig. 19. Similarly, the digital device stores a sample from the noise distribution (1404), and this sample is uploaded to the analog device by charging the capacitors appropriately (1408). With the connectivity chosen and the capacitors charged, the analog device is ready to evolve in time under the reverse process (either the Reverse SDE or Reverse ODE) (1410). This evolution involves continuous communication with the pre-trained score network (1412), which is stored on the digital device as shown in Fig. 12. At the end of the evolution, the data is downloaded onto the digital device by reading off the voltages across the capacitors on the analog device and then passing the resulting voltages through an ADC (1414). Finally, if appropriate, the data is decoded (e.g., mapped from a latent space to the original feature space) (1416).

Fig. 15 shows a flowchart for the scenario where the score network is trained by the user. In this case, the user generates (noisy) training data by evolving under the forward process and stores the training data on the digital device (1502). Initially the data may be encoded on the digital device (e.g., into a latent space or a spectral domain) (1504). The data is then uploaded to the analog device by appropriately charging the capacitors of the unit cells (1510). The problem geometry is stored on the

digital device (1402) and uploaded to the analog device by choosing the connectivity of the unit cells (1406) as in the pre-trained case. With the connectivity chosen and the capacitors charged, the analog device then evolves over time (1518) according to the so-called forward process (forward SDE or forward ODE), which adds noise to the data. The noisy data is then downloaded to the digital device by reading off the capacitors' voltages and passing them through an ADC (1520). This data acts as training data for the score matching optimization, with the loss function given in Eq. (8). This results in a trained score network, which is stored on the digital device (1512). At this point we now have access to a trained score network, which means that we are at the same point as the starting point of the algorithm in Fig. 14. This means that the rest of the protocol follows the same steps as the protocol in Fig. 14.

14 Computational advantages (assuming digital score network)

The are advantages of performing the forward and reverse processes in analog hardware with respect to doing them digitally. These are:

1. SDEs are in general numerically unstable for large dimensions, requiring sophisticated numerical integrators and fine-tuned time-stepping schedules. With an analog system, these difficulties disappear, as there is no time step to be chosen.

2. The total physical time can be tuned, and depends on the operating decay rate of the system, proportional to $1/(RC)$ for a single unit cell. This means the physical time of integration can be made faster, an advantage of analog computation.

3. For the forward process, the voltages should be initialized to match the input data $\mathbf{x}$ and then to measure the voltages for each unit cell at each instant of a chosen discretization for training the score network. For the reverse process, no intermediate measurements are needed, and the initial voltages are set to match the noisy data $\mathbf{x}(T)$. In both cases, this does not involve matrix multplications and inversions that are present in the SDE, since the hardware already obeys the SDE. Digitally, for the general case (in particular when $\mathbf{G}$ is a matrix), these operations would have to be performed, costing many more steps of $O(N^2)$ operations.

15 Analog Score Network

15.1 Weaknesses of a digital score network

One issue associated with a digital score network is the issue of latency, which leads to a time delay, as described immediately below.

During the evolution of the reverse process, a feedback loop occurs involving the score network. Namely, at each step in which the reverse process evolves in time, the score network is queried for the predicted score $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, t)$. Querying a digital score network takes time, because it involves a forward pass through a deep neural network. This forward pass involves multiple matrix-vector multiplications and non-linear activation functions. Hence, it takes some non-trivial amount of time to perform this forward pass and to receive the predicted value of the score $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, t)$. There is also an additional delay coming from the DACs and ADCs by which the state variable $\mathbf{x}$ is converted to a digital signal and the predicted score is converted to an analog signal. We refer to the time-delay associated with this process as the *latency* of the score network.

When the reverse process is performed on a digital device, a numerical SDE solver discretizes time and digitally integrates the reverse process. In this context, the latency of the score network implies

that the numerical SDE solver should be slowed down. In other words, each time step of the numerical integration may take longer, due to this latency, since every single time step involves a forward pass through the score network.

When the reverse process is performed on an analog device, such as the device described above, the latency of the score network is also an issue. In this case, the reverse process is evolving continuously on an analog device. Because this system is evolving continuously in a physical system, the evolution cannot be interrupted or slowed down, once it starts. This implies that the score values received by the reverse process may not be fully accurate—they may be time delayed. So whenever the reverse process expects a value of the score $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, t)$ at time t, it would actually receive the score $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, t+\tau)$ at some delayed time or shifted time $t+\tau$. (Note that since t decreases over time during the reverse process, a time delay corresponds to a positive shift in t.) In other words, the latency of the score network translates into inaccurate values of the score. In turn this may translate into inaccurate time evolution of the state variable $\mathbf{x}$, which could affect the quality of the samples that are generated by the generative model. Of course, if these inaccuracies are intolerable,the physical parameters of the reverse process can be chosen to slow down the reverse process, trading speed for accuracy. If the time delay caused by the digital evaluation of the score network is small enough, this approach may still be useful.

In summary, for digital diffusion processes, the latency of the score network translates into a slowing down of the sample generation process. On the other hand, for analog diffusion processes, it translates into either into inaccuracies in the time evolution or into a slowing down of the sample generation process, or both.

This motivates removing the latency issue with the *analog* version of the score network disclosed below.

15.2 General strategy for removing latency

Our general strategy for removing latency is to have an analog device that acts as score network. We refer to this as the score device.

The score device evolves in real time, simultaneous with the time evolution of the reverse process. Specifically, the score devices evolves in from time $\tau = 0$ to time $\tau = T_f$, where τ denotes the real (physical) time. At each time point, the score device outputs a prediction for the score.

A subtle point is that there could be some transformation $f(\tau)$ that relates the real time value τ to the abstract time value $t = f(\tau)$. Here t is the abstract time associated with the reverse diffusion process. The reverse process evolves in such a way that the abstract parameter t evolves from $t = T$ to $t = 0$. Typically, we take $f(\tau) = T - t$, so that τ evolves from $0 \to T$ while t evolves from $T \to 0$. When the real time τ reaches some time $\tau = \tau_0$, the score device outputs a predicted score of $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, f(\tau_0))$, and in practice we can choose f such that $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, f(\tau_0)) = \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, T - \tau_0)$.

The fact that the score device and the reverse process evolve simultaneously with each other allows us to address the latency issue, since the two devices can communicate with each other in real time, without relying on the intervention of a digital device. Overall, the two devices evolve together according to a system of differential equations, where the two differential equations are coupled.

The simultaneous time evolution concept is useful even if one of the devices is digital. For example, it is beneficial if the reverse diffusion process is digitally solved with a numerical SDE solver while the analog score network evolves in time. It is also beneficial if the reverse diffusion process is analog but the score network is digitally integrated with an ODE solver.

However, the benefit is likely more pronouced if both devices are analog. In this case, the two devices can be physically coupled with an analog link, and all signals can remain analog (i.e., no analog-to-digital conversion is necessary). Hence, in what follows, we focus mostly on the case where both the score device

and the reverse process correspond to analog devices.

15.3 General differential equations for the analog system

Assuming the reverse process corresponds to an analog device as described above, the reverse process evolves according to:

$$d\mathbf{v}(t) = \Big(\mathbf{L}(t)\mathbf{v} - \mathbf{M}(t)\mathbf{M}(t)^{\intercal}\mathbf{s}(\mathbf{v}, t)\Big)dt + \mathbf{M}(t)\mathrm{d}\overline{\mathbf{w}}_t \tag{62}$$

Here and henceforth we use the notation $\mathbf{s}(\mathbf{v}, t) = \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{v}, t)$ to denote the output of the trained score network, i.e., the output associated with the parameters $\boldsymbol{\theta}^*$ obtained after training the score network.

Note that t is an abstract variable and we can make the transformation to physical time with the change of variables $\tau = T - t$, leading to:

$$d\mathbf{v}(T-\tau) = -\Big(\mathbf{L}(T-\tau)\mathbf{v} - \mathbf{M}(T-\tau)\mathbf{M}(T-\tau)^{\intercal}\mathbf{s}(\mathbf{v}, T-\tau)\Big)d\tau + \mathbf{M}(T-\tau)\mathrm{d}\mathbf{w}_\tau \tag{63}$$

Here, we used the fact that $dt = -d\tau$. Also, we used the fact that $\mathrm{d}\overline{\mathbf{w}}_t = \mathrm{d}\mathbf{w}_\tau$, since the overline on $\mathrm{d}\overline{\mathbf{w}}_t$ is used when the time variable evolves with negative increments (as t does), and this overline can be removed when the time variable evolves with positive increments (as τ does).

The score device evolves according to its own differential equation. The differential equation for the score can be written as follows:

$$\frac{d\mathbf{s}}{d\tau} = \mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \frac{d\mathbf{v}}{d\tau}, \mathbf{s}) \tag{64}$$

where $\mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \frac{d\mathbf{v}}{d\tau}, \mathbf{s})$ is some function of τ, $\mathbf{v}$, $\frac{d\mathbf{v}}{d\tau}$, and $\mathbf{s}$, and this function depends on trainable parameters denoted by $\boldsymbol{\theta}$. We allow for flexibility in the precise form of the function $\mathbf{h}_{\boldsymbol{\theta}}$, although we give some possible example forms for this function below. In fact, $\mathbf{h}_{\boldsymbol{\theta}}$ could even have a stochastic component, i.e., it could either be a deterministic or stochastic function. *Taken together, Equations (63) and (64) form a system of coupled differential equations that simultaneously evolve forward in real time τ.*

15.4 General integral equations for the score device

This section pertains to the score predicted during the reverse diffusion process.

The actual values of the score predicted by the score network depend both on the differential equation as well as the initial condition. The predicted score values have the form:

$$\mathbf{s}(\mathbf{v}, t = T - \tau') = \mathbf{s}_0 + \int_{\tau=0}^{\tau=\tau'} \frac{d\mathbf{s}}{d\tau}d\tau = \mathbf{s}_0 + \int_{\tau=0}^{\tau=\tau'} \mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \frac{d\mathbf{v}}{d\tau}, \mathbf{s})d\tau \tag{65}$$

In this equation, $\mathbf{s}_0 = \mathbf{s}(\mathbf{v}_0, \tau = 0)$ is the initial condition associated with the dynamics, corresponding to the score value at $\tau = 0$ for an initial vector $\mathbf{v}_0 = \mathbf{v}(\tau = 0)$. Assuming the relationship $t = T - \tau$, we can also think of the initial condition $\mathbf{s}_0$ as being the score value $\mathbf{s}(\mathbf{v}(t = T), t = T)$, i.e., the score at the abstract time $t = T$. As discussed in Sec. 9, the distribution at the abstract time $t = T$ corresponds to the so-called noisy distribution, p_{noise}. Hence the initial condition $\mathbf{s}_0$ is related to the score of p_{noise}.

Oftentimes, p_{noise} is simple enough such that we may have an analytical description of the score function. (For example, p_{noise} is often a multi-variate Gaussian distribution with a known mean an covariance.) Hence, in some cases, we can assume that the functional form of the term $\mathbf{s}_0$ is known ahead of time, prior to running the diffusion model.

In other cases, we may not know the precise form of the initial condition, but we can learn this function through the score matching process described in Sec. 3.2. In this case, we can think of the initial condition on the score as a free parameter that should be learned, and hence we can write the

initial condition as $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}, t = T)$, to indicate that it is parameterized. We can therefore write the predicted score values, for some set of parameters $\boldsymbol{\theta}$, as:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}, t = T - \tau') = \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}, t = T) + \int_{\tau=0}^{\tau=\tau'} \mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \frac{d\mathbf{v}}{d\tau}, \mathbf{s}) d\tau \tag{66}$$

Typically we can go a step further and argue that the initial condition is an affine function, of the form $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}, t = T) = A_{\boldsymbol{\theta}}\mathbf{v} + \mathbf{b}_{\boldsymbol{\theta}}$ where $A_{\boldsymbol{\theta}}$ is a parameterized matrix and $\mathbf{b}_{\boldsymbol{\theta}}$ is a parameterized vector. In other words, for an initial vector $\mathbf{v}_0$, we can write the initial condition as:

$$\mathbf{s}_0 = \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}_0, t = T) = A_{\boldsymbol{\theta}}\mathbf{v}_0 + \mathbf{b}_{\boldsymbol{\theta}} \tag{67}$$

The argument for why the condition at $\tau = 0$ (or $t = T$) is approximately affine is as follows. For the forward diffusion process, the final distribution as $t \to \infty$, denoted p_{noise}, is unique and independent of the initial data distribution p_{data}. This means that we can choose an arbitrary initial distribution at $t = 0$ (i.e., an arbitrary data distribution p_{data}) for the forward process, and still end up at the same final distribution as $t \to \infty$. For example, we could choose a data distribution that is peaked on a single datapoint $\mathbf{x}(0)$. In this case, the stationary distribution is a multi-variate Gaussian distribution (for the forward processes that we consider, which have an affine drift term), since in this case the stationary distribution corresponds to a conditional distribution, of the form $p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))$. This means that, in general for arbitrary p_{data}, the corresponding stationary distribution p_{noise} is a multi-variate Gaussian distribution. Hence, assuming that T is large, then the distribution at $t = T$ will approximately correspond to a multi-variate Gaussian distribution. The score of a Gaussian is an affine function, and hence this is why we can assume that the condition at $\tau = 0$ (which is approximately the score of p_{noise}) is affine.

Combining Equations (66) and (67) gives the formula for the parameterized output of the score device:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{v}, t = T - \tau') = A_{\boldsymbol{\theta}}\mathbf{v}_0 + \mathbf{b}_{\boldsymbol{\theta}} + \int_{\tau=0}^{\tau=\tau'} \mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \frac{d\mathbf{v}}{d\tau}, \mathbf{s}) d\tau \tag{68}$$

There are trainable parameters inside of $A_{\boldsymbol{\theta}}$, $\mathbf{b}_{\boldsymbol{\theta}}$, and $\mathbf{h}_{\boldsymbol{\theta}}$.

15.5 Example differential equations for the score device based on total derivative

Consider two different examples for how one can construct the score device.

Example 1: In the first example, we consider constructing the score device based on the theoretical concept of the total derivative. The score $\mathbf{s}(\mathbf{v}(\tau), \tau)$ is a function of both τ and $\mathbf{v}(\tau)$. Because of this, we can write the total derivative with respect to τ as follows:

$$\frac{d\mathbf{s}}{d\tau} = \frac{\partial \mathbf{s}}{\partial \tau} + \sum_i \frac{\partial \mathbf{s}}{\partial \mathbf{v}_i} \frac{d\mathbf{v}_i}{d\tau} = \frac{\partial \mathbf{s}}{\partial \tau} + (\nabla_{\mathbf{v}} \mathbf{s}) \cdot \left(\frac{d\mathbf{v}}{d\tau} \right) \tag{69}$$

Here $\nabla_{\mathbf{v}}$ denotes the gradient with respect to the $\mathbf{v}$ vector. Copmaring the form of Equation (69) with the form of Equation (64) shows that the expression $\frac{\partial \mathbf{s}}{\partial \tau} + (\nabla_{\mathbf{v}} \mathbf{s}) \cdot \left(\frac{d\mathbf{v}}{d\tau} \right)$ provides a blueprint for how to construct the function $\mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \mathbf{s})$.

Viewing the score as a static function $\mathbf{s}(\mathbf{v}, \tau)$, the form of this static function is determined by: (1) the data distribution p_{data} and (2) the forward diffusion process (i.e., the forward SDE). However, we can also view the score as something that follows a trajectory, if we imagine that τ and $\mathbf{v}(\tau)$ are progressing with time. In the latter case, the trajectory has an initial condition and then evolves dynamically over

time. From the trajectory perspective, the evolution of the score depends on the trajectory associated with $\mathbf{v}(\tau)$ via the derivative term $\frac{d\mathbf{v}}{d\tau}$ that appears in Equation (83). However, this term $\frac{d\mathbf{v}}{d\tau}$ can be independent of the static form of the score function. Hence, if training a score network to predict the value of the score, then this model does not need to predict $\frac{d\mathbf{v}}{d\tau}$. Rather, this term can be viewed as an *input* to the model.

This implies that we can develop a model for the score function by rewriting Equation (83) as follows

$$\frac{d\mathbf{s}}{d\tau} = \mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau) + \mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau) \cdot \frac{d\mathbf{v}}{d\tau} \tag{70}$$

with

$$\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau) \approx \frac{\partial \mathbf{s}}{\partial \tau} \quad \text{and} \quad \mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau) \approx \nabla_{\mathbf{v}} \mathbf{s}. \tag{71}$$

In other words, the trainable function $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ provides a model for the partial derivative $\frac{\partial \mathbf{s}}{\partial \tau}$, and the trainable function $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ provides a model for the gradient $\nabla_{\mathbf{v}} \mathbf{s}$. This constitutes a natural splitting of the $\mathbf{h}_{\boldsymbol{\theta}}$ function into two functions $\mathbf{q}_{\boldsymbol{\theta}}$ and $\mathbf{r}_{\boldsymbol{\theta}}$ that describe different contributions to the time derivative of the score function.

The form of the functions $\mathbf{q}_{\boldsymbol{\theta}}$ and $\mathbf{r}_{\boldsymbol{\theta}}$ are flexible, although some intuition is helpful. The fraction $\frac{\partial \mathbf{s}}{\partial \tau}$ can be thought of as related to the forward diffusion process, as it describes how the score changes with time during that process. Therefore, this term may involve drift and diffusion processes, similar to that in the forward process. On the other hand, the gradient $\nabla_{\mathbf{v}} \mathbf{s}$ is especially reflective of the data distribution p_{data}.

These two functions may be quite complex can be constructed using a neural-network inspired approach. Sec. 15.6 below discloses a suitable circuit architecture for such a neural-network inspired approach.

Once the training process is over such that the parameters $\boldsymbol{\theta}$ are fixed, then the score device can be used in conjunction with the reverse diffusion process. After training, we can denote the functions $\mathbf{q}_{\boldsymbol{\theta}}$ and $\mathbf{r}_{\boldsymbol{\theta}}$ as $\mathbf{q} = \mathbf{q}_{\boldsymbol{\theta}^*}$ and $\mathbf{r} = \mathbf{r}_{\boldsymbol{\theta}^*}$, respectively, where $\boldsymbol{\theta}^*$ corresponds to the parameters after training. Because we will be using the score device in conjunction with the reverse process, we can then substitute in the formula for $\frac{d\mathbf{v}}{d\tau}$ associated with the reverse process. In particular, we can use the formula in Equation (83), which gives:

$$\frac{d\mathbf{s}}{d\tau} = \mathbf{q}(\mathbf{v}, \tau) + \mathbf{r}(\mathbf{v}, \tau) \cdot (-\mathbf{L}(T-\tau)\mathbf{v} + \mathbf{M}(T-\tau)\mathbf{M}(T-\tau)^{\intercal}\mathbf{s}(\mathbf{v}, T-\tau) + \mathbf{M}(T-\tau)\mathbf{w}) \tag{72}$$

We use $\mathbf{w}$ here since we previously defined $\mathbf{w}dt = d\mathbf{w}$, and also we emphasize that this is the same $\mathbf{w}$ (the same source of stochastic noise) as that appearing in the reverse diffusion process. In summary, Equation (72) represents the differential equation for how the score evolves over time during the reverse process, assuming the total-derivative model presented in this section.

Eq. (72) is a formal mathematical statement. In practice, one can export the value of $\frac{d\mathbf{v}}{d\tau}$ from the reverse process to the score device and explicitly use the formula in Eq. (145) for the physical device. For example, each component of $\frac{d\mathbf{v}}{d\tau}$ may physically correspond to (or be proportional to) the voltage across a resistor that is in series with the capacitor in each unit cell of the reverse diffusion process. Hence, the voltage across each of these resistors is an instantaneous measurement of $\frac{d\mathbf{v}}{d\tau}$ and can be fed to the score device.

15.6 Circuit architecture for total derivative approach

Figure 18 is an overall schematic diagram for an example score device 340, which includes six (sets of) components for the total derivative approach:

1. voltage sources 341 whose outputs are $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$;

2. voltage mixers 342 that multiply the components of $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ with the corresponding components of $\frac{d\mathbf{v}}{d\tau}$, which are physically encoded in voltage vectors;

3. a voltage source 343 whose output is $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$;

4. voltage adders 344 that add the voltages from the voltage sources together to produce overall voltage signals $\mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \frac{d\mathbf{v}}{d\tau}) = \mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau) + \mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau) \cdot \frac{d\mathbf{v}}{d\tau}$;

5. integrator circuits 345 that output voltages $\mathbf{i}_{\boldsymbol{\theta}}(\tau')$ that are the time integrals of the signals $\mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \frac{d\mathbf{v}}{d\tau})$, of the form $\mathbf{i}_{\boldsymbol{\theta}}(\tau') = \int_{\tau=0}^{\tau=\tau'} \mathbf{h}_{\boldsymbol{\theta}}(\tau, \mathbf{v}, \frac{d\mathbf{v}}{d\tau}) d\tau$; and

6. voltage adders 346 that add the output voltages of the integrators, $\mathbf{i}_{\boldsymbol{\theta}}(\tau')$, with the initial condition $\mathbf{s}_0$ from voltage source 347, to produce an output voltage 348 corresponding to the predicted score value $\mathbf{s}_{\boldsymbol{\theta}}(\tau') = s_0 + \mathbf{i}_{\boldsymbol{\theta}}(\tau')$.

Figure 16 depicts several circuits as black boxes. This includes the adder circuits 344 and 347, the integrator circuits 345, and the circuits 341 and 343 associated with the voltage sources $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ and $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$. We elaborate on the form of these voltage sources in the next subsection.

Figure 17 shows examples of an adder circuit (left) and an integrator circuit (right) suitable for use in the score device 340.

15.7 Form of the voltage sources $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ and $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$

Recall that the score function $\mathbf{s}_\theta(\mathbf{v}, t)$ is an N-dimensional vector. Hence the model $\mathbf{q}_\theta(\mathbf{v}, t)$ is also an N-dimensional vector that is trained to approximate the partial time derivative of the score function. Moreover, each component $s_\theta^{(i)}$ of the score function has an N-dimensional gradient (with respect to $\mathbf{v}$), and for each i, we train another model $r_\theta^{(i)}$ to approximate this gradient. We define $r_\theta(\mathbf{v}, t)$ as a matrix whose columns are given by vectors $r_\theta^{(i)}$.

The models $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ and $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ contain the neural network parameters that have to be trained. As mentioned previously, an issue arises if these outputs are evaluated digitally (for example, coming out of a neural network programmed on a CPU); there will be latency in coming from the evaluation itself, as well as the digital-to-analog conversion (as well as analog-to-digital), possibly hindering performance.

We can therefore construct fully analog neural networks for $\mathbf{q}$ and each of the $r^{(i)}$. Figure 23 shows a general analog system for evaluating $\mathbf{q}$ or any of the $\mathbf{r}^{(i)}$ at positions $\mathbf{v}$ and times $t > 0$. As the score device is coupled to a trajectory $\mathbf{v}(t)$ of the forward process, we evaluate $\mathbf{q}$ and $\mathbf{r}^{(i)}$ at all space-time points $(\mathbf{v}(t), t)$ along the trajectory. These values are ported into each of the networks as continuous voltage signals, where the components of $\mathbf{v}(t)$ are directly picked off the device running the forward process. The time t is continuously ported into the analog network as a ramp voltage with unit slope that begins rising from a base of zero when the coupled forward trajectory begins; this ensures that when the network receives $\mathbf{v}(t)$ it also receives the time at which these readings occurred. In what follows, we describe the analog network in terms of the N valued functions $\mathbf{r}_{\boldsymbol{\theta}}^{(i)}$, but the network architecture also applies to the $\mathbf{q}_{\boldsymbol{\theta}}$ function.

Each network receives $N + 1$ voltage signals $(\mathbf{v}(t), t)$ as input, and the analog neural architecture performs a series of layered transformations. Each layer is denoted by the pair $\{A, B\}_k$; A_k is a block implementing an parameterized affine transformation, and this is followed by an element-wise nonlinear transformation B_k. The number of layers K and the width of each layer M_k are left as hyperparameters. Figure 23 shows the electrical system that performs these layered transformations.

The first affine sublayer A_1 is implemented by passing the voltage signals $(\mathbf{v}(t), t)$ through a network of resistors; for now, we limit to the case that the affine transformation is just a matrix multiplication (No added bias term). This is shown in Figure 24.

To obtain the dot product of the matrix A_1 with the input, we create M_1 copies of $(\mathbf{v}(t), t)$ and use each to compute a dot product with one of the M_1 rows of A_1. The copying procedure amounts to splitting each wire carrying a component of $(\mathbf{v}(t), t)$ into M branches according to Figure 24 thanks to Kirchhoff's second law. To compute the dot product between a copy of $(\mathbf{v}, t)$ and the jth row of A_1, we attach a resistor in series to each of wires carrying a component of the copy, and then join all wires together at a node after the resistors. The voltage at the node is the dot product of $(\mathbf{v}(t), t)$ and the vector of inverse resistance values (conductances) attached to each wire. If each wire carrying a voltage component signal $(\mathbf{v}, t)^{(i)}$ is placed in series with a resistor with inverse resistance equal to the (i, j)th entry of A_1, the voltage reading at the node where all wires meet will be the dot product between $(\mathbf{v}, t)$ and the jth row of A_1. If we have M_1 copies of $(\mathbf{v}(t), t)$, we can repeat this construction for each row of A_k, and output each component of the vector $A_1(\mathbf{v}(t), t)$ as M voltage signals. The same idea applies to each layer A_k.

The previous construction assumes the current exiting each layer, from the nodes after the resistors, is zero. This assumption can be realized in practice by inserting a voltage follower as a buffer between each summing node and the subsequent layer. An example voltage follower is shown in Figure 25.

An alternative construction is possible using a summing amplifier. Consider again the voltage copying procedure above and the example of multiplying a copy of $(\mathbf{v}, t)$ and the j^{th} row of A_1. We show a circuit that performs this dot product in Figure 26.

We prove the dot product property as follows. First, the high input impedance of the op amp means we can assume the current into the inputs of the op amp is zero. This lets us use Kirchoff's current law at node x to write

$$I_R = \sum_{i=1}^{d+1} I_i \qquad \longrightarrow \qquad \frac{V_x - V_{\text{out}}}{R} = \sum_{i=1}^{d+1} (A_1)_{j,i} \, (v_i(t) - V_x). \tag{73}$$

Next, since the op amp is used in closed loop operation (there is a path from the output to the inverting input), we can additionally assume the voltages at the inverting and non-inverting inputs are equal. Thus $V_x = 0$, so that we have

$$V_{\text{out}} = -R \sum_{i=1}^{d+1} (A_1)_{j,i} \, v_i(t) = -R \, (A_1)_j \cdot (\mathbf{v}, t). \tag{74}$$

To get a pure dot product, we can set $R = 1$ and invert the output.

The nonlinear layer B_k can be constructed by attaching a diode followed by a resistor after each linear layer A_k, shown in Fig. 23. Measuring the voltage at each resistor yields a nonlinearly transformed voltage thanks to the nonlinearity of the characteristic function of the diode (or any other electrical nonlinear element). The transformed voltage is therefore of the form $\mathbf{v}' = R f_d(\mathbf{v})$, with f_d the characteristic function of the diode. The current-voltage charactertistic of diode under forward bias resembles the activation function associated with a rectified linear unit (ReLU). This provides motivation for using a forward-biased diode, since ReLU is one of the most common activation functions used in neural networks.

The output of this analog neural network is an N-dimensional voltage vector, corresponding to either $\mathbf{r}_{\boldsymbol{\theta}}^{(i)}(\mathbf{v}, t)$ or $\mathbf{q}_{\boldsymbol{\theta}}$, that is fed into voltage mixers or adders, as explained in above and depicted in Fig. 16. Ideally, the outputs $\mathbf{r}_{\boldsymbol{\theta}}^{(i)}(\mathbf{v}, t)$ and $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, t)$ reach the mixers and adders at exactly time t, although it could take some non-zero time for a pass through the analog neural network. However, the electric fields

generated by the input voltage signals propagate through the analog components of the neural network at the speed of light, so the output is computed almost instantaneously.

The parameters $\boldsymbol{\theta}$ are the values of tunable resistors used throughout the affine layers of the network. In the spirit of digital neural networks, the nonlinear layers with diodes have no tunable components. The resistors can be tuned to reduce or minimize a cost function $\mathcal{L}$, whose evaluation can be done in the analog domain as explained in the next subsection.

Another possibility is to consider an analytical form for $\mathbf{q}$ and $\mathbf{r}$, whose parameters would be coefficients of a chosen series expansion. These would have the general form:

$$\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v},\tau) = \sum_i \theta_i^{(1)} J_i(\alpha^{(1)},\mathbf{v},\tau) \tag{75}$$

$$\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v},\tau) = \sum_i \theta_i^{(2)} J_i(\alpha^{(2)},\mathbf{v},\tau) \tag{76}$$

where J_i are basis functions, $\alpha^{(1)}, \alpha^{(2)}$ are hyperparameters and $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}$ are sets of parameters to learn. One example of such basis functions are the generalized Bessel functions, defined by functions

$$J_i^{\mathbf{p}}(\mathbf{x}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} \exp\left[i\left(n\xi - \sum_k^m x_k \sin p_k k\xi\right)\right] d\xi \tag{77}$$

for an N-dimensional input vector $\mathbf{x}$, where $\mathbf{p} = (p_1,\ldots,p_N)$ are additional parameters to train. By using such forms for $\mathbf{q}$ and $\mathbf{r}$, they can be calculated analytically or at low numerical cost and fed in analogically to the voltage mixer. This is motivated by the fact that the overall voltage signal $\mathbf{h}_\theta$ may be complex, but is ultimately composed of simpler terms $\mathbf{q}_\theta(\mathbf{v},\tau) \approx \frac{\partial \mathbf{s}}{\partial \tau}$ and $\mathbf{r}_\theta(\mathbf{v},\tau) \approx \nabla_{\mathbf{v}}\mathbf{s}$ which may be amenable to analytical forms.

Another solution in between a fully analog and digital evaluation of the score function is using an FPGA for inference, where $\mathbf{q}_\theta(\mathbf{v},\tau)$ and $\mathbf{r}_\theta(\mathbf{v},\tau)$ are neural networks programmed on an FPGA. One advantage of such an approach is that all the advantages of digital inference using CPUs and GPUs such as parallelization may be kept while having a low latency to feed into the voltage mixer to obtain $\mathbf{h}$.

15.8 Alternative score device based on a simple circuit

Example 2: In the second example, we consider constructing the score device based on a simple circuit. In this case, the score device is composed of the following elements.

Figure 22A shows a circuit diagram for a single unit cell. Here we break up the resistor into two resistors R_i and ρ_i, since the resistors become inequivalent when we introduce the coupling between the unit cell.

Figure 22B shows a circuit diagram for two unit cells coupled together, via a capacitor. This is essentially the same kind of coupling we introduced for the forward diffusion process.

1. The score device is composed of N unit cells.

2. Each unit cell contains three components in series: (1) a capacitor with capacitance $C_{\theta,i}$, (2) a resistor with resistance $R_{\theta,i}$, (3) a voltage source $V_{\theta,i}(\mathbf{x})$ whose output depends on the input vector $\mathbf{x}$.

3. All three of these components are parameterized by parameters (hence the subscript θ) that can be trained during the training process.

4. The N unit cells are capacitively coupled to each other according to some connectivity matrix. This coupling can follow the same geometry as that used by the forward diffusion process, as described in Sec. 7 and Fig. 10.

5. The voltage source $V_{\theta,i}(\mathbf{x})$ is essentially an analog neural network, involving multiple layers, including affine layers and non-linear activation functions.

For the connectivity between the unit cells, we can use the same strategy as that employed in Fig. 10, which involves switches that can be toggled off or on based on the connectivity matrix. In fact, we can choose the exact same connectivity matrix for the score device as that employed in the forward diffusion process.

The voltage source $V_{\theta,i}(\mathbf{x})$ can have a very similar structure as that of the $\mathbf{q}$ and $\mathbf{r}$ sources discussed in Sec. 15.7. The main difference is that $V_{\theta,i}(\mathbf{x})$ does not explicitly depend on the time parameter t, and hence the trainable weights are time independent. Other than that difference, the structure is similar to that of $\mathbf{q}$ and $\mathbf{r}$, in that $V_{\theta,i}(\mathbf{x})$ is composed of alternating layers of affine functions, followed by non-linear activation functions.

15.9 Differential equations for the alternative score device

Next, we present the time evolution for the alternative score device discussed in the previous subsection within our general framework. Recall that Eq. (84) gives the general form of the differential equation. Our alternative score device is a relatively simple model in that the $\mathbf{h}_{\boldsymbol{\theta}}$ function has no explicit dependence on either τ or $\frac{d\mathbf{v}}{d\tau}$. Thus, in this case, we can write

$$\frac{d\mathbf{s}}{d\tau} = \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{v}, \mathbf{s}). \tag{78}$$

We can define the capacitance matrix $\mathbf{C}_\theta$ in a manner similar to Eq. (47), and the resistance matrix as $\mathbf{R}_\theta \equiv \text{diag}(R_{\theta,1}, R_{\theta,2}, ..., R_{\theta,N})$. Then, we arrive at the following differential equation for the entire score device:

$$\frac{d\mathbf{s}}{d\tau} = \mathbf{C}_\theta^{-1}\mathbf{R}_\theta^{-1}(\mathbf{s} + \mathbf{V}_\theta(\mathbf{v})) \tag{79}$$

and hence in this case $\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{v}, \mathbf{s}) = \mathbf{C}_\theta^{-1}\mathbf{R}_\theta^{-1}(\mathbf{s} + \mathbf{V}_\theta(\mathbf{v}))$.

The intuition for why this differential equation is potentially useful is that it captures some of the intuition of the total derivative approach. Because Eq. (79) includes both a time derivative term $\frac{d\mathbf{s}}{d\tau}$ and a term proportional to $\mathbf{s}$, this implies that changes in the state variable $\mathbf{v}$ affect both the time derivative of the score as well as the score itself. This is a similar intuition to the total derivative approach. In addition, a non-trivial dependence of the score on the time τ comes from the differential equation itself, which has the geometric notions encoded in the matrix $\mathbf{C}_\theta$. Hence, the time dependence of $\mathbf{s}$ can be non-trivial via the evolution of Eq. (79).

This approach could also be turned into a hybrid digital-analog score device. This would involve using a digital device, such as an FPGA, to produce the voltage source $\mathbf{V}_\theta(\mathbf{v})$. In other words, $\mathbf{V}_\theta(\mathbf{v})$ can be represented by a digital neural network. A DAC can convert the output of this neural network to an analog voltage, which is applied to the analog unit cells in the score device, and the analog unit cells the analog score used by the reverse diffusion process.

In what follows, we focus primarily on the total derivative approach in Sec. 15.5. However, the formalism we develop can also be applied to the alternative score device presented here.

15.10 Incorporating problem geometry and inductive bias into the analog score device

We have presented two approaches to constructing an analog score network (or hybrid digital-analog score network).

In practice, both of these approaches benefit from accounting for problem geometry. Sec. 7 and Fig. 9 show that different problems have different geometries. Accounting for this geometry improves performance in many machine learning tasks, including generative modeling.

In the context of analog score devices, problem geometry can be accounting for in the circuits used to implement the parameterized voltage sources. Specifically, this refers to the sources $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ and $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ in the device based on the total derivative, and the source $\mathbf{V}_{\theta}(\mathbf{v})$ in the alternative score device based on the simple circuit.

These voltage sources can be modeled as analog neural networks where each layer performs an affine transformation followed by a non-linear activation function. The affine transformation is represented by a matrix A, as discussed in Sec. 15.7 and depicted in Fig. 24. The problem geometry can be incorporated into each of the A matrices associated with these affine transformations. For example, these A matrices can be sparse, with non-zero elements only corresponding to the connectivity of the problem. In other words, these A matrices can be chosen to have a similar structure to the adjacency matrix $\mathcal{A}$ from Sec. 7. By matching the zero elements of the A matrices to the zero elements of the adjacency matrix $\mathcal{A}$, this provides a recipe for incorporating problem geometry into the analog voltage sources $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$, $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$, and $\mathbf{V}_{\theta}(\mathbf{v})$, and hence into the analog score devices.

Physically speaking, this corresponds to limiting the number of wires uses in the circuit structure in Fig. 24. In other words, if the adjacency matrix $\mathcal{A}$ has non-zero elements associated with the wires in Fig. 24, the wires can be bundled together. Hence, the adjacency matrix can be used to guide the construction of the circuit in Fig. 24, to incorporate problem geometry.

While these remarks apply to the case when $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$, $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$, and $\mathbf{V}_{\theta}(\mathbf{v})$ are analog devices, problem geometry can be incorporated into digital devices, and hence the overall score device is hybrid digital-analog. In this case, standard techniques can be used to incorporate inductive biases into digital neural networks.

15.11 Evaluating the loss function based on the score of the conditional distribution

The loss function can be written in various forms. Here we will discuss the loss function in Eq. (9), and then discuss an alternative loss function in the next subsection.

The loss function in Eq. (9) is written in terms of the conditional distributions $p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))$, as follows:

$$\mathcal{L}_1(\boldsymbol{\theta}) = \mathbb{E}_t \left\{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)} \left[\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t)|\mathbf{x}(0)) \|_2^2 \right] \right\}. \tag{80}$$

This loss function uses knowledge of the scores of the conditional distributions.

In some special cases, we may have an analytical formula for the scores of the conditional distributions. Namely, when the forward SDE is particularly simple, the conditional distributions will also be simple. The scores of the conditional distributions are affine functions whenever the drift term in the SDE is affine. However, in general, the form of this affine function might not be known. One particular case where we can analytically solve for the score of the conditional distribution is when the matrix $\mathbf{L}(t)$, which is defined in Eq. (31), is a diagonal matrix.

Consider the case where we know the analytical form for the scores of the conditional distributions,

and this form is affine. In this case we can write:

$$\nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t)|\mathbf{x}(0)) = A(t)\mathbf{x}(t) + \mathbf{b}(t) \tag{81}$$

where we assume that we know $A(t)$ and $\mathbf{b}(t)$. (When we do not have an analytical form like this, we can use the approach in the next subsection instead.)

We will now describe the various steps involve in evaluating the above loss fucntion.

(1) Generating the score of the conditional distribution (SCD) as a voltage:

We can now discuss how to generate the SCD as an analog voltage.

The vector $\mathbf{v}(t)$ is a voltage vector that corresponds to the data vector $\mathbf{x}(t)$. This vector $\mathbf{v}(t)$ is produced by the forward process evolving up to time t. Hence, $\mathbf{v}(t)$ can be stored on the analog device. On the other hand, the matrix $A(t)$ and vector $\mathbf{b}(t)$ can be stored digitally, e.g., on memory of a CPU. This provides two options for how to compute the SCD value:

$$\mathbf{s}_{\mathbf{x}(t)|\mathbf{x}(0)} = A(t)\mathbf{v}(t) + \mathbf{b}(t) \tag{82}$$

A first option is to send the vector $\mathbf{v}(t)$ through an analog-to-digital converter, and then compute Eq. (82) on a digital device (the same device that stores $A(t)$ and $\mathbf{b}(t)$). The resulting vector can then be fed back to the analog device with a digital-to-analog converter.

A second option is to compute Eq. (82) on the analog device. This involves treating $\mathbf{b}(t)$ as a voltage vector and using a voltage adder to add this vector with $A(t)\mathbf{v}(t)$. The computation of the latter could be done, for example, by encoding the elements of A in the resistances of resistors and then using a circuit similar to the adder circuit, which computes a weighted average of the elements in $\mathbf{v}$.

Hence, we assume that one of these options are chosen, and as a result, we have the vector $\mathbf{s}_{\mathbf{x}(t)|\mathbf{x}(0)}$ stored on the analog device as a voltage vector.

(2) Obtaining predictions $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$ for the score value

Next, consider how to obtain the value of $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$ also as a voltage vector. Consider a particular time point of interest $t = t_m$, where $0 \leq t_m \leq T$. We can write:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(T), T) = \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t_m), t_m) + \int_{t=t_m}^{t=T} \frac{d\mathbf{s}}{dt} dt \tag{83}$$

This is just a statement that the score at the final time T is equal to the score at a prior time t_m, plus the integral of the time derivative of the score from t_m to T.

We are considering the forward process now, and hence we are using the time variable t, instead of the time variable τ, which we typically write as $\tau = T - t$. Note that $\partial t = -\partial\tau$, and hence

$$\frac{\partial \mathbf{s}}{\partial t} = -\frac{\partial \mathbf{s}}{\partial \tau} \tag{84}$$

Hence, in the total derivative model, if we have $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}(\tau), \tau) \approx \frac{\partial \mathbf{s}}{\partial \tau}$, then we have

$$-\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}(T-t), T-t) \approx \frac{\partial \mathbf{s}}{\partial t} \tag{85}$$

So, in terms of t, the total derivative model looks like this:

$$\frac{d\mathbf{s}}{dt} = -\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}(T-t), T-t) + \mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}(T-t), T-t) \cdot \frac{d\mathbf{v}}{dt} \tag{86}$$

In any case, let us rewrite Eq. (88) as follows:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t_m), t_m) = \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(T), T) - \int_{t=t_m}^{t=T} \frac{d\mathbf{s}}{dt} dt \tag{87}$$

We can compute the desired score value $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t_m), t_m)$ if we know the score at the final time T. As discussed previously, the distribution at time T is essentially the noise distribution p_{noise}. In particular, in Sec. 18.4, we discuss that p_{noise} can be assumed to be Gaussian, in which case its score is an affine function. We write this affine function as:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(T), T) = A_{\boldsymbol{\theta}}\mathbf{x}(T) + \mathbf{b}_{\boldsymbol{\theta}} \tag{88}$$

where $A_{\boldsymbol{\theta}}$ and $\mathbf{b}_{\boldsymbol{\theta}}$ can be viewed as parameterized, trainable objects. Hence we obtain the result:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t_m), t_m) = A_{\boldsymbol{\theta}}\mathbf{x}(T) + \mathbf{b}_{\boldsymbol{\theta}} + \int_{t=t_m}^{t=T} \mathbf{h}_{\boldsymbol{\theta}}(T - t, \mathbf{v}, -\frac{d\mathbf{v}}{dt}, \mathbf{s}) dt \tag{89}$$

where we made the substitution $\frac{d\mathbf{s}}{dt} = -\frac{d\mathbf{s}}{d\tau} = -\mathbf{h}_{\boldsymbol{\theta}}(T - t, \mathbf{v}, -\frac{d\mathbf{v}}{dt}, \mathbf{s})$.

Equation (89) provides a blueprint for how to obtain score values from our model, in order to evaluate the loss function. Namely, we follow the following process.

1. Initialize the analog forward process with a fixed datapoint $\mathbf{x}(0)$;
2. Evolve the analog forward process in time from $t = 0$ to $t = t_m$;
3. At time $t = t_m$, initialize the analog score device based on the values of the variables at time t_m;
4. Evolve forward both the analog score device and the analog forward process simultaneously, from $t = t_m$ to $t = T$, while computing the integral of $\mathbf{h}_\theta$ that appears in Eq. (89), which is output as a voltage vector;
5. At time T, obtain the vector $x(T)$, which is the final result of the forward process, and use it to compute $A_{\boldsymbol{\theta}}\mathbf{x}(T) + \mathbf{b}_{\boldsymbol{\theta}}$ as a voltage vector; and
6. On the analog device, add together the two voltage vectors from the two previous steps (Step 5 and Step 6) to obtain the resulting score value $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t_m), t_m)$ as a voltage vector.

(3) Taking the difference of the score values

By now, we have described how to obtain both $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$ and $\mathbf{s}_{\mathbf{x}(t)|\mathbf{x}(0)}$ as voltage vectors. Both of these voltage vectors are defined relative to ground. Hence, if we want to compute their difference, we can simply pick off the relative voltage between them. That is, if we want the vector

$$\mathbf{d}(t) = \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) - \mathbf{s}_{\mathbf{x}(t)|\mathbf{x}(0)} \tag{90}$$

then we pick off the voltage of $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$ relative to the voltage of $\mathbf{s}_{\mathbf{x}(t)|\mathbf{x}(0)}$.

(4) Computing the norm squared of a voltage vector

The next step in computing the loss function is to compute the norm squared of the vector $\mathbf{d}(t)$. This can be done, up to a proportionality constant, using the circuit in Fig. 18. We feed $\mathbf{d}(t)$ into this circuit, and the output voltage is proportional to the norm squared:

$$N(t) = \|\mathbf{d}(t)\|^2 \tag{91}$$

(5) Computing the expectation values in the loss function

The final step is to compute the three expectation values appearing in Eq. (9). The first expectation value $\mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)}$ involves sampling different $\mathbf{x}(t)$ from a fixed starting point of $\mathbf{x}(0)$. Operationally, this involves running the forward process (with a fixed starting point) multiple times, say, K times, and then averaging over all runs. Let $N_{kl}(t_m) = \|\mathbf{d_{kl}}(t_m)\|^2$ denote the norm squared of the score difference in Eq. (90) for the kth run of the forward process, with a fixed starting point $\mathbf{x}(0)_l$, where the forward process is run up to a time t_m. Then we can estimate the expectation value $\mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)}$ with the estimator E_{lm}, given by:

$$E_{lm} = \frac{1}{K}\sum_{k=1}^{K} N_{kl}(t_m) \tag{92}$$

In practice, it may be easiest to compute Eq. (92) digitally. This involves taking the voltage $N_{kl}(t_m)$, passing it through an analog-to-digital converter, and then computing the average in (92) on a digital device, such as a CPU. Hence, we assume that E_{lm} is stored on a digital device.

The next step is to compute the expectation value $\mathbb{E}_{\mathbf{x}(0)}$. The digital device produces samples of $\mathbf{x}(0)$ from the data distribution p_{data}. Hence, we estimate this expectation value with an estimator

$$F_m = \frac{1}{L}\sum_{l=1}^{L} E_{lm} \tag{93}$$

This involves producing L samples of $\mathbf{x}(0)$ on the digital device, and computing the average in Eq. (92) on the digital since, since E_{lm} is already stored digitally.

Finally, we compute the expectation value $\mathbb{E}_t$. This may involve a weighting function $\lambda(t)$ (which can be stored digitally). Once again, we can estimate this expectation value on the digital device. We sample t_m uniformly over the time interval $[0, T]$ and introduce the estimator:

$$G = \frac{1}{M}\sum_{m=1}^{M} \lambda(t_m) F_m \tag{94}$$

This estimator can be straightforwardly computed on the digital device.

G is an unbiased estimator for the loss function $\mathcal{L}_1$ in (88). This means that the expectation value of G is equal to $\mathcal{L}_1$. The implications are that this leads to convergence guarantees for optimizers that employ G as an estimator, since convergence guarantees often rely on the estimator being unbiased. This is relevant to the optimization discussion in Sec. 15.14.

15.12 Evaluating the loss function based on the trace of the Hessian

An alternative loss function based on the trace of the Hessian of the log probability can be written as

$$\mathcal{L}_2(\boldsymbol{\theta}) = \mathbb{E}_t\left\{\lambda(t)\mathbb{E}_{\mathbf{x}(0)}\mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)}\left[\frac{1}{2}\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t)\|_2^2 + \mathrm{Tr}[\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t)]\right]\right\}. \tag{95}$$

In the context of our total derivative model for the score device, this loss function is especially natural to evaluate. This is because our total derivative model constructs an explicit model for the Hessian of the log probability. The function $\mathbf{r}_{\boldsymbol{\theta}}$ is, in fact, a model for the Hessian of the log probability. That is, we have:

$$\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{x}(t), T-t) = \nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t) \tag{96}$$

Hence we can rewrite Eq. (95) as

$$\mathcal{L}_2(\boldsymbol{\theta}) = \mathbb{E}_t\left\{\lambda(t)\mathbb{E}_{\mathbf{x}(0)}\mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)}\left[\frac{1}{2}\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t)\|_2^2 + \mathrm{Tr}[\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{x}(t),T-t)]\right]\right\}. \tag{97}$$

Let us now discuss how to evaluate this loss function. Let $\mathbf{r}_{\boldsymbol{\theta},j}^{(i)}$ denote the (i,j)th entry of the $\mathbf{r}_{\boldsymbol{\theta}}$ matrix. Then we have:

$$\beta(t,\mathbf{x}(t),\mathbf{x}(0)) := \mathrm{Tr}[\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{x}(t),T-t)] = \sum_i \mathbf{r}_{\boldsymbol{\theta},i}^{(i)}(\mathbf{x}(t),T-t) \tag{98}$$

Computing the term in Eq. (98) then just involves the following:

1. Do a forward pass on each function $\mathbf{r}_{\boldsymbol{\theta},i}^{(i)}$ for the voltage vector input $\mathbf{v} = \mathbf{x}(t)$ and time input $\tau = T - t$. This forward pass uses the construction for $\mathbf{r}_{\boldsymbol{\theta}}$ based on the discussion in Sec. 15.7.

2. Take the voltages $\mathbf{r}_{\boldsymbol{\theta},i}^{(i)}$ from the previous step and add them with a voltage adder circuit. The result is an output voltage $\beta(t,\mathbf{x}(t),\mathbf{x}(0))$.

Next let us discuss how to compute the term

$$\alpha(t,\mathbf{x}(t),\mathbf{x}(0)) := \frac{1}{2}\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t)\|_2^2 \tag{99}$$

In the previous subsection, we gave a protocol for how obtain score value outputs from our model, based on the formula in Eq. (89). Hence, we follow the following steps:

1. Follow the protocol in the previous subsection to obtain score values $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t)$ via the formula in Eq. (89).

2. Take the norm squared of this voltage vector using an analog circuit, such as the one in Fig. 15.

3. An appropriate proportionality factor can be added to the output of the previous step via a voltage amplifier. The result is an output voltage $\alpha(t,\mathbf{x}(t),\mathbf{x}(0))$.

The next step is to add these voltages, e.g., on the analog device, to obtain a voltage given by:

$$\gamma(t,\mathbf{x}(t),\mathbf{x}(0)) = \alpha(t,\mathbf{x}(t),\mathbf{x}(0)) + \beta(t,\mathbf{x}(t),\mathbf{x}(0)) \tag{100}$$

The final step is to compute the expectation values in Eq. (95). This can be done digitally, which involves first converting the analog voltage $\gamma(t,\mathbf{x}(t),\mathbf{x}(0))$ into a digital signal, with an analog-to-digital converter. One can then follow a procedure very similar to that described in the previous subsection. Namely, we introduce an estimator for the loss function in (95), as follows:

$$G = \frac{1}{M}\frac{1}{L}\frac{1}{K}\sum_{m=1}^{M}\sum_{l=1}^{L}\sum_{k=1}^{K}\lambda(t_m)\gamma(t_m,\mathbf{x}(t_m)_l,\mathbf{x}(0)_k). \tag{101}$$

Here, $\mathbf{x}(0)_l$ denotes the lth sampling of $\mathbf{x}(0)$ from p_{data}, and $\mathbf{x}(t_m)_{kl}$ denotes the lth sampling of $\mathbf{x}(t_m)$ from running the forward process up to time t_m starting from initial point $\mathbf{x}(0)_l$. Eq. (101) can be computed on the digital device. Moreover, G provides an unbiased estimator of the loss function in Eq. (95) and hence has desireable convergence properties when used inside of an optimization routine.

15.13 Evaluating alternative loss function with analog time integration

In the loss functions discussed above, evaluating the expectation values is done digitally and does not take full advantage of the analog device. In particular, analog devices can efficiently compute time integrals, and hence the analog device could be useful for evaluating an expectation value $\mathbb{E}_t$ over time t.

For this purpose, we consider alternative loss functions that involve first computing the expectation value $\mathbb{E}_t$, instead of doing it last. Namely, we pull $\mathbb{E}_t$ inside the mathematical formula such that it occurs

before the other expectation values, $\mathbb{E}_{\mathbf{x}(0)}$ and $\mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)}$. In general, this can alter the loss function, i.e., swapping the order of these expectation values can actually change the loss function. Nevertheless, the intuition of the score matching process seems to still be captured even when the expectation values are reversed, and hence the new loss function may still be useful for score matching.

This strategy can be used for both of the previously considered loss functions, i.e., those in Eq. (86) and Eq. (95). When adapting Eq. (86) in this way, we arrive at:

$$\mathcal{L}_3(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}(0)}\mathbb{E}_{\mathcal{T}|\mathbf{x}(0)}\mathbb{E}_t\left[\lambda(t)\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t) - \nabla_{\mathbf{x}(t)}\log p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))\|_2^2\right]. \tag{102}$$

Here $\mathcal{T}$ denotes a trajectory of the forward process, and $\mathcal{T}|\mathbf{x}(0)$ denotes such a trajectory given an initial starting point of $\mathbf{x}(0)$. Hence $\mathbb{E}_{\mathcal{T}|\mathbf{x}(0)}$ denotes the expectation value of all trajectories of the forward process, given an initial starting point of $\mathbf{x}(0)$.

In Sec. 15.11, we outlined a protocol for obtaining the argument of the norm in this loss function. That is, we provided a protocol to obtain the value of $\mathbf{d}(t)$ as an analog voltage, where

$$\mathbf{d}(t) = \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t) - \mathbf{s}_{\mathbf{x}(t)|\mathbf{x}(0)}\,. \tag{103}$$

Encoding $\sqrt{\lambda(t)}$ as an analog voltage and using a voltage mixer to multiply this by $\mathbf{d}(t)$ yields a voltage vector:

$$\tilde{\mathbf{d}}(t) = \sqrt{\lambda(t)}\mathbf{d}(t)\,. \tag{104}$$

The next step then is to compute the expectation value $\mathbb{E}_t$ of $\|\tilde{\mathbf{d}}(t)\|_2^2$. There is an efficient subroutine for this using an analog device. Specifically, analog devices are capable of computing the time integral of the square of a voltage.

We consider the thermal method for RMS (root-mean-square) voltage measurement. In this method, an unknown, time-dependent voltage heats a resistor R_1. A digital device (e.g., CPU) applies DC voltage to an equal resistance R_2, until both resistors reach the same temperature. The DC voltage is then equal to the RMS voltage of the unknown source. The temperature sensing can be carried out by two semiconductor diodes, which can be viewed as thermistors. The overall circuit diagram is shown in Fig. 18, and also includes an op amp. The idea is to apply this subroutine to the voltage source $\tilde{\mathbf{d}}_i(t)$, which is the ith component of the voltage vector $\tilde{\mathbf{d}}(t)$. In that case, the output of the subroutine would be

$$a_i = \sqrt{\mathbb{E}_t(\tilde{\mathbf{d}}_i(t))^2}\,. \tag{105}$$

Next we apply the subroutine in Fig. 18 to the vector $\mathbf{a} = \{a_i\}$, which returns the norm squared of the vector, giving an output voltage proportional to:

$$\sum_i a_i^2 = \mathbb{E}_t(\tilde{\mathbf{d}}(t))^2 \tag{106}$$

An analog-to-digital converter converts this analog voltage to a digital value. Next, this same procedure is repeated for many different runs of the forward process., and the results are averaged to estimate the expectation value $\mathbb{E}_{\mathcal{T}|\mathbf{x}(0)}$. The expectation value $\mathbb{E}_{\mathbf{x}(0)}$ is obtaind by averaging over $\mathbf{x}(0)$ values.

Similarly, the loss function in Eq. (95) can be adapted to obtain the new loss function:

$$\mathcal{L}_4(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}(0)}\mathbb{E}_{\mathcal{T}|\mathbf{x}(0)}\mathbb{E}_t\left[\lambda(t)\{\frac{1}{2}\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t)\|_2^2 + \mathrm{Tr}[\nabla_{\mathbf{x}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t)]\}\right]. \tag{107}$$

The time expectation value $\mathbb{E}_t$ of the norm squared term can be evaluated on the analog device using the thermal method describe above, around Eq. (103). On the other hand, the time expectation value $\mathbb{E}_t$ of

the trace term can be evaluated on the analog device using the a standard integrator circuit. The two contributions from the two terms can be summed together, and the rest of the computation associated with this loss function can be done straightforwardly.

15.14 Training the parameters of the score device

Fig. 20 illustrates a hybrid analog-digital feedback loop 2000 for training the parameters of the score device 340. In this feedback loop 2000, the analog device evaluates the loss function (or its gradient) for a fixed value of the parameters θ (2210). The result of this evaluation is sent to the digital device. The digital device chooses a new value for the parameters θ', based on some optimization routine (2220). (Examples of such optimization routines could be gradient descent, stochastic gradient descent, or gradient-free methods like Nelder-Mead). The new values θ' for the parameters are then programmed into the analog device in order to evaluate the loss function (or its gradient) again.

This feedback loop 2000 can be iterated multiple times until some convergence criterion is reached. For example, the convergence criterion could say that the optimization terminates when the loss function fails to decrease substantially in value for several iterations in a row. We denote the final value of the parameters, after the convergence criterion is met, as θ^*.

This final set of parameters θ^* can be uploaded onto the score device for use in the reverse process. In other words, after the training process, we fix the parameters of the score device to be θ^*, and use these parameters throughout the evolution of the reverse process.

15.15 Feeding analog signals to the reverse diffusion process

Figure 27 shows the communication process between the reverse diffusion process and either an analog score network or a hybrid digital-analog score network. For simplicity, this is shown for the case of a single unit cell ($N = 1$). Figure 27 shows two possible unit cell constructions are considered: a unit cell based on voltage mixers and a unit cell based on variable resistors. The reason for the complicated circuits is that the score value should be multiplied by a prefactor of $g(t)^2$ where $g(t)$ is the prefactor on the diffusion term.

16 Flowchart of entire system with analog score network

Figure 21 shows a flowchart of the entire thermodynamic AI system, whenever both the diffusion process (forward and reverse process) as well as the score network all correspond to analog devices. This involves a hybrid analog-digital optimization loop for the training process, with analog and digital devices evaluating a loss function (2010) and the digital device performing an optimization routine (2020) to provide the trained score network (2112) as illustrated in Fig. 20. In addition, after training, the analog score device interfaces with the reverse process, providing score values that are used by the reverse process. This flowchart can be compared to the flowchart in Fig. 15, which illustrates the case of a digital score network. Overall the two flowcharts are similar, although there are more steps performed by analog components in Fig. 21, which can provide additional computational advantages, as discussed immediately below.

17 Computational advantages with analog or hybrid analog-digital score network

If the full device is analog, i.e., the diffusion device where each RC cell represents a data element, and the score device is fully analog, we retain the computational advantages presented in 14; solving the reverse

SDE on a digital device involves very numerical expensive algorithms like the Euler-Maruyama method. As such, computing the reverse SDE solution in a time interval $0 < \tau < T$ can take much longer than T seconds if done on a digital computer. Moreover, computation time scales poorly as the number of dimensions of the reverse SDE is increased. On the other hand, our analog SDE integrator can solve the reverse SDE in the same interval in exactly T seconds. Moreover, the computation time should remain T seconds regardless of the SDE's dimension.

With the speed advantage of analog SDE integration established, we show how an analog score network ensures the solutions the reverse process provided by an analog integrator are accurate. The analog score network addresses the issue of latency in continuous trajectories of the reverse process. To be clear, the latency issue encountered during the analog reverse process is not an issue for a digital device simulating the same process because a digital system solves the reverse SDE with discrete time steps, and the time to query the score network does not affect the discrete solver's solution.

Solving the latency issue ensures that the reverse process can be solved accurately with our analog SDE integrator. In short, the reverse process can be solved much faster on an analog device, and the analog score network can ensure such solutions rival the accuracy of digital solutions that take much longer to obtain. Specifically, the analog system solving the reverse process gains the following advantages when it includes an analog score network:

- The drift term of the reverse process need not be computed with a digital score network. Hence, there is no latency from the digital evaluation of the score network with a general neural network. (see Fig. 8). As such, the drift term in the reverse process can more accurately follow its true shape along a trajectory.

- No latency from the DACs to be used at each time step of the forward and reverse processes. The only remaining DACs are those used for the initialization, where a data vector $\mathbf{x}$ is converted to a voltage vector $\mathbf{v}$, and the collection of $\mathbf{v}$ at the end of the time evolution.

- In the case where the derivative of the h function is split into the $\mathbf{q}$ and $\mathbf{r}$ function, see section 15.5, the size of the analog score device device may be small, (have fewer variational parameters) since this approach is based on splitting the terms that contribute to the derivative of the score function. Splitting terms results in tracking simpler contributions, which may use fewer resources.

- By using an analog score network, we can solve for the score along a continuous trajectory of points, where these continuous evaluations may be ported into another analog device which evaluates loss functions against it. (depending on its exact form, see section 15.13. This may be less computationally expensive than a digital evaluation of the loss function that involves many evaluations of a digital neural network representing the score, and vastly increases the number of score evaluations used as data when evaluating the loss function.

18 Deep learning and neural networks

Deep learning (DL) and neural networks (NNs) are employed in language translation, product recommendations, social media, dynamic pricing, and other applications. Deep learning refers to employing artificial neural networks as trainable mathematical models, where the number of layers in the neural network is often large and hence the network is deep. A deep learning system extracts high level features about a dataset that are useful for classifying the data in the dataset. Prototypical example applications of deep learning are classifying images of handwritten digits or classifying images of cats and dogs. In practice, deep learning has extremely broad applications in many fields.

The left panel of Figure 29 gives a simple illustration of a neural network. Here an input variable x is mapped to an output y, via a hidden layer. (Usually multiple hidden layers are used.) Trainable weights determine the mapping from x to the hidden layer variables $\{h_j\}$ and the mapping from the hidden layer variables to the output y. The hidden layer variables represent high-level features of the data, to enhance classification performance.

19 Overconfidence in deep learning

Machine learning systems such as neural networks are often overconfident in their predictions. For low-stakes applications, this may not be a major issue. However, some applications, such as self-driving cars and medical diagnosis, are higher stakes in the sense that making a wrong decision can lead to consequences relevant to human life. Overconfidence can be catastrophic for these high-stakes applications.

At a technical level, this overconfidence often arises because neural networks are trained on limited amounts of training data. These training data points live in a vast feature space. Hence it is common for some regions of feature space to be poorly represented by the training data, i.e., these regions may be far away from the training data. When it comes time to test the trained neural network on testing data, the testing data could be in a region that is far away from training data, and yet the neural network will still attempt to make a prediction. Because the neural network is not familiar with these regions, the neural network is not aware that it should be careful when making predictions for them.

20 Uncertainty quantification in deep learning

One strategy for dealing with overconfidence is uncertainty quantification (UQ). UQ aims to quantify the uncertainty of the predictions made by the neural network. UQ is useful for high-stakes applications (e.g., cancer detection in medicine) because it provides guidance for when the user should defer to human judgement over the machine's predictions. UQ is widely recognized as making machine learning more reliable and trustworthy.

Several different methods exist for UQ in machine learning. A simple example of UQ is adding confidence intervals to the predictions made by the neural network.

A more sophisticated and rigorous approach to UQ is the Bayesian framework. The Bayesian framework quantifies uncertainty by accounting for prior knowledge (often called the prior distribution) and updates that knowledge due to data or observations (often called the posterior distribution). Bayesian methods aim to quantitatively capture knowledge in the form of probability distributions.

21 Bayesian neural networks (BNNs)

Neural networks are machine learning models that typically have multiple layers of linear and non-linear transformations, allowing them to express a wide variety of potential functions. Bayesian neural networks (BNNs) allow for uncertainty quantification on the predicted outputs of the neural network. This improves the reliability and trustworthiness of the model.

Figure 29 illustrates differences between a Bayesian neural network (right) and a standard neural network (left). In the Bayesian case, the weights do not have definite values. Instead the weights are randomly drawn from a particular probability distribution.

Instead of predicting a definite y for a given input x, a BNN predicts y according to some probability distribution. Hence there is randomness in the output prediction for a given input. Specifically, BNNs

make probabilistic predictions based on the following formula:

$$p(y|x, D) = \int_w p(y|x, w)p(w|D)dw, \tag{108}$$

where $D = \{x^{(m)}, y^{(m)}\}$ is the training data, w are the weights (i.e., parameters) of the BNN, x represents a test input data point, and y represents the output. Here, $p(y|x, w)$ is the predictive distribution for a given value of the weights w, and $p(w|D)$ is the posterior distribution on the weights after training with data D.

22 Challenges of large-scale Bayesian deep learning

The posterior distribution $p(w|D)$ for the weights of a BNN is high-dimensional and non-convex and often has multiple modes. Consequently, computing Eq. (108) involves doing an integral over a multi-million dimensional multi-modal posterior, with unusual topological properties like mode-connectivity. Exact methods for sampling from the posterior distribution can take thousands of training epochs to produce a single sample from the posterior. In practice, computing Eq. (108) with exact methods is computationally infeasible with modern computing methods.

Hence, for computational reasons, researchers approximate the posterior distribution using computationally inexpensive methods. For example, researchers often use mini-batch methods that only use a subset of the data. However, these methods can provide heavily biased estimates of the true posterior expectation in Eq. (108). The mean-field approximation is another approach, the true posterior distribution is approximated by a single-mode Gaussian distribution, although this approach can likewise be a poor approximation of Eq. (108). For illustration, a multimodal distribution is shown in Fig. 30. Fitting this multimodal distribution to a single-mode Gaussian distribution would be a poor approximation.

In conclusion, obtaining high-accuracy predictions from BNNs at large scale remains an unsolved problem. It is possible to obtain crude approximations of these predictions with reasonable computational speed or high-accuracy predictions at an extremely (prohibitively) slow rate. However, there is currently no way to accurately compute Eq. (108) with fast computational speed at large scale.

23 Technical details and subroutines of Bayesian deep learning

Here we provide additional technical details about the various components, subsystems, and subroutines used in Bayesian deep learning. Specifically, we focus on an approach to Bayesian deep learning called stochastic differential equations Bayesian neural networks (SDE-BNNs). The SDE-BNN approach is a state-of-the-art method for Bayesian deep learning, performing competitively with or better than other methods. The SDE-BNN approach is somewhat inspired by physics and be implemented on digital hardware.

23.1 Variational Inference

Bayesian inference involves updating a prior distribution to a posterior distribution based on data or observations. Variational inference (VI) is a form of Bayesian inference that involves postulating an ansatz $\mathcal{Q}$ (i.e., a family of possible solutions) for the posterior distribution. Optimizing over the distributions $q(w)$ in the ansatz $\mathcal{Q}$ yields an approximate posterior distribution. This optimization problem can be written as:

$$q^*(w) = \arg\min_{q(w)\in\mathcal{Q}} D_{KL}(q(w)||p(w|D)), \tag{109}$$

where D_{KL} is the Kullback-Leibler (KL) divergence and $p(w|D)$ is the true probability of the weight values w given the data D.

23.2 The Evidence Lower Bound (ELBO)

Minimizing or reducing the KL divergence is equivalent to maximizing or increasing the so-called evidence, which is given by $\log p(D) = \log \int p(D, w) dw$. The integral in the evidence formula is typically intractable to compute, in some cases taking exponential time to compute. Therefore, it is common to maximize or increase a more tractable quantity, which lower bounds the evidence.This quantity is called the Evidence Lower BOund (ELBO), $\mathcal{L}_{ELBO} \leq \log p(D)$, and has the following form:

$$\mathcal{L}_{ELBO} = (\mathbb{E}_{q(w)} \log p(D|w)) - D_{KL}(q(w)||p(w)). \tag{110}$$

Hence, the optimization problem becomes the following:

$$q^*(w) = \arg \max_{q(w) \in \mathcal{Q}} \mathcal{L}_{ELBO} \tag{111}$$

23.3 Stochastic Variational Inference

Stochastic Variational Inference (SVI) combines natural gradients with stochastic optimization. The idea behind SVI is to use a cheaply computed, noisy, unbiased estimate of the natural gradient inside of a gradient ascent optimization. The natural gradient is different from the standard, Euclidean gradient, as it accounts for the geometry of the problem. SVI works with an unbiased estimator of the natural gradient, rather than the natural gradient itself.

23.4 Neural ODEs

Neural Ordinary Differential Equations (ODEs) represent continuous depth versions of artificial neural networks. The values of the hidden units in artificial neural networks represented by neural ODEs are denoted h_t and the values of the weights are denoted w_t. In general, both of these quantities depend on time, and evolve according to the coupled differential equations:

$$\frac{d}{dt} \begin{bmatrix} h_t \\ w_t \end{bmatrix} = \begin{bmatrix} f_h(t, h_t, w_t) \\ f_w(t, w_t) \end{bmatrix}.$$

Hence, a forward pass through the neural network involves integrating this system of differential equations.

23.5 Neural SDEs

Neural Stochastic Differential Equations (SDEs) represent a continuous-depth version of Bayesian neural networks. Once again, a continuous-depth version of a BNN evolves according to a system of coupled differential equations. However, the system is stochastic in nature:

$$d \begin{bmatrix} h_t \\ w_t \end{bmatrix} = \begin{bmatrix} f_h(t, h_t, w_t) \\ f_w(t, w_t) \end{bmatrix} dt + \begin{bmatrix} \mathbf{0} \\ g_w(t, w_t) \end{bmatrix} dB, \tag{112}$$

where dB is a Brownian motion term.

23.6 SDE for Prior Distribution

The prior distribution can be formulated over the weights of a BNN as an SDE. Specifically, the SDE can take the forms of the function $f_w(t, w_t)$ and $g_w(t, w_t)$ that appear in Eq. (112).

As an example, consider the Ornstein–Uhlenbeck (OU) process. For the OU process, $f_w(t, w_t) = -w_t$ and $g_w(t, w_t) = \sigma I_d$. Hence, for the prior distribution, the SDE system is:

$$d\begin{bmatrix} h_t \\ w_t \end{bmatrix} = \begin{bmatrix} f_h(t, h_t, w_t) \\ -w_t \end{bmatrix} dt + \begin{bmatrix} \mathbf{0} \\ \sigma I_d \end{bmatrix} dB \tag{113}$$

23.7 SDE for Posterior Distribution

The posterior distribution can also be formulated over the weights of a BNN as an SDE. The posterior distribution should be highly expressive. Hence, the SDE can be more complicated than the SDE of a prior distribution. We therefore choose the drift term for the weights to be described by a neural network NN_ϕ with trainable parameters ϕ. Specifically, we can choose the following SDE for the posterior distribution:

$$d\begin{bmatrix} h_t \\ w_t \end{bmatrix} = \begin{bmatrix} f_h(t, h_t, w_t) \\ \mathrm{NN}_\phi(t, w_t, \phi) + w_t \end{bmatrix} dt + \begin{bmatrix} \mathbf{0} \\ \sigma I_d \end{bmatrix} dB \tag{114}$$

The trainable parameters in this model include both the parameters ϕ appearing in the drift term as well as the initial condition w_0 on the weights. The neural network NN_ϕ can be referred to as the Posterior Drift Network (PDN), since it determines the drift associated with the posterior distribution.

23.8 Simplified expression for ELBO

Whenever the SDEs for the prior and posterior distributions have the same diffusion terms, then the ELBO expression can be simplified. In this case we have

$$\mathcal{L}_{ELBO} = \mathbb{E}_{q(w)}\left[\log p(D|w) - \frac{1}{2}\int_0^1 \|u(t, w_t, \phi)\|_2^2 dt\right] \tag{115}$$

where the function u is given by

$$u(t, w_t, \phi) = g_w(t, w_t)^{-1}[f_w^p(t, w_t) - f_w^q(t, w_t, \phi)]. \tag{116}$$

Here, f_w^p is the drift associated with the prior, and f_w^q is the drift associated with the approximate posterior.

23.9 Computational difficulties with a digital approach to SDE-BNNs

In practice, digital processors are used to train and store SDE-BNNs. However, digital approaches to SDE-BNNs face certain computational difficulties, including:

1. large amounts of training data for training the Posterior Drift Network;
2. high complexity for training and evaluating the Posterior Drift Network;
3. long times for numerically simulating the dynamics of the ODEs and SDEs; and
4. potential instability of SDEs/ODEs due to large dimensionality and stiffness, leading to impractically small time steps.

The first two difficulties above largely stem from a lack of inductive bias in the problem setup. Inductive bias refers to prior knowledge being inputted into the structure of the model, for example knowledge of symmetries in the data. Having a strong inductive bias can reduce the training data requirements as well as improve the speed of training of the model.

The third difficulty refers to the challenge of using digital hardware to simulate time dynamics. This includes both the challenge of digitally generating Gaussian randomness (i.e., the d**B** term in the SDE) and the challenge of numerical time integration via discretizing the time dynamics. Finally, the fourth difficulty refers to the fact that there are many cases in which numerically integrating SDEs and ODEs may lead to instabilities due to the structure of the equations, particularly for SDEs when the number of dimensions is large. Another issue is known as stiffness, which leads to smaller time steps and can make numerical integration impractical.

24 A Thermodynamic AI System for Bayesian Deep Learning

Figure 33 illustrates a thermodynamic AI system for Bayesian deep learning with subsystems that can perform four subroutines. Each subsystem can be implemented as a physical analog device (subsystem/component) that performs a corresponding subroutine. Some subset of the subroutines may also be stored in and processed on a digital device. These four subsystems include:

1. a Weight Diffuser (WD) 3310;
2. a Hidden Layer Network (HLN) 3320;
3. a Posterior Drift Network (PDN) 3330; and
4. a Loss Evaluator (LE) 3340.

At a high level, Figure 33 illustrates how these four subsystems interact with each other. The Weight Diffuser (WD), which can represent both the prior distribution and the posterior distribution, feeds weight values to the Hidden Layer Network (HLN). The WD also communicates back-and-forth with the Posterior Drift Network (PDN): the WD feeds weight values to the PDN and the PDN feeds drift values to the WD. The Loss Evaluator (LE) takes in signals from all three of the other subsystems—the HLN, the WD, and the PDN—in order to evaluate the loss function.

25 Hidden Layer Network

25.1 Overview of HLN

The Hidden Layer Network (HLN) is represented by the differential equation:

$$\frac{dh_t}{dt} = f_h(t, h_t, w_t). \tag{117}$$

The output of the HLN is given by the integral equation:

$$h_1 = h_0 + \int_0^1 f_h(t, h_t, w_t)dt. \tag{118}$$

Here, h_0 is typically set to some input data value. For example, during the training process with training data $D = \{x^{(m)}, y^{(m)}\}$, one would choose $h_0 = x^{(m)}$ for some value of the index m. After HLN training is complete, the HLN can make predictions on some test input x for $h_0 = x$.

The HLN can be viewed as a neural ordinary differential equation (neural ODE). This neural ODE can be stored and processed on a digital device or can be implemented on an analog device. In what follows we discuss both the digital case and the analog case. We also discuss how one can augment the model with additional dimensions to improve performance. Finally, we discuss how to obtain predictions for output values y from the HLN.

25.2 Digital Hidden Layer Network

One possible setup for the overall system is for the HLN to be stored and processed on a digital device, while the other devices (WD, PDN, and LE) have analog components. The digital device that stores and processes the HLN could be a central processing unit (CPU) or field programmable gate array (FPGA).

This setup involves conversion between digital and analog signals. For example, there can be an analog-to-digital converter that converts the weight values (outputted by the WD) from analog to digital signals, so that the weights can be processed by the HLN. There could also be a digital-to-analog converters, e.g., when converting the output predictions made by the HLN from digital to analog signals.

One technological benefit of this approach is the notion of drop-in uncertainty quantification (drop-in UQ). The name drop-in UQ is inspired by the idea of providing a non-invasive service whereby uncertainty quantification is added as a feature on top of an existing (e.g., digital) architecture.

In industrial applications, it is often the case that a user has their own HLN and their own dataset stored on a digital device. In this case, a physical device can offer a drop-in UQ service where uncertainty quantification is provided to the user's application. This would involve interfacing the user's HLN and dataset with our physical device (which includes the WD, PDN, and LE).

25.3 Analog Hidden Layer Network

An HLN can be implemented in analog hardware as follows. Consider a dataset $D = \{x^{(m)}, y^{(m)}\}_{m=1}^{M}$ composed of M datapoints. Suppose that each input vector $x^{(m)} = \{x_j^{(m)}\}_{j=1}^{N}$ is N-dimensional, meaning that there are N features associated with each datapoint.

(1) The Unit Cell

An analog HLN can have N unit cells, corresponding to one unit cell for each data feature. Figure 32 shows a possible architecture for a unit cell in the HLN. The voltage across the capacitor represents the state variable h_t. When doing a circuit analysis, the resistor in the unit cell provides the time derivative term $\frac{dh_t}{dt}$, the capacitor and voltage source α_j provide the linear drift term (linear in h_t), and the non-linear element (NLE), such as a diode or transistor, provides a non-linear drift term.

(2) Two Coupled Unit Cells

Figure 33 shows two unit cells for the HLN that are coupled together via a resistive bridge. The following differential equations represent the voltages, v_j and $v_{j'}$, across the capacitors:

$$-C_j \frac{dv_j}{dt} = \left(\frac{1}{R_j} + \frac{1}{R_{jj'}}\right) v_j - \frac{v_{j'}}{R_{jj'}} - \frac{\alpha_j}{R_j} + I_{\mathrm{NL},j} \tag{119}$$

$$-C_{j'} \frac{dv_{j'}}{dt} = \left(\frac{1}{R_{j'}} + \frac{1}{R_{jj'}}\right) v_{j'} - \frac{v_j}{R_{jj'}} - \frac{\alpha_{j'}}{R_{j'}} + I_{\mathrm{NL},j'}. \tag{120}$$

Here $I_{\mathrm{NL},j}$ and $I_{\mathrm{NL},j'}$ are the respective currents in the non-linear element branches of the jth and j'th unit cells. The matrix form of this system of equations is then:

$$-\mathbf{C}\dot{\mathbf{v}} = \mathbf{J}\mathbf{v} - \mathbf{J}_{\mathrm{s}}\alpha + \mathbf{I}_{\mathrm{NL}}, \tag{121}$$

where

$$\dot{\mathbf{v}} \equiv \begin{bmatrix} \frac{dv_j}{dt} \\ \frac{dv_{j'}}{dt} \end{bmatrix}, \ \mathbf{C} \equiv \begin{bmatrix} C_j & 0 \\ 0 & C_{j'} \end{bmatrix}, \ \mathbf{J} \equiv \begin{bmatrix} \frac{1}{R_j} + \frac{1}{R_{jj'}} & -\frac{1}{R_{jj'}} \\ -\frac{1}{R_{jj'}} & \frac{1}{R_{j'}} + \frac{1}{R_{jj'}} \end{bmatrix}, \ \mathbf{J}_{\mathrm{s}} \equiv \begin{bmatrix} \frac{1}{R_j} & 0 \\ 0 & \frac{1}{R_{j'}} \end{bmatrix},$$
$$\mathbf{v} \equiv \begin{bmatrix} v_j \\ v_{j'} \end{bmatrix}, \ \alpha \equiv \begin{bmatrix} \alpha_j \\ \alpha_{j'} \end{bmatrix}, \text{ and } \mathbf{I}_{\mathrm{NL}} \equiv \begin{bmatrix} I_{\mathrm{NL},j} \\ I_{\mathrm{NL},j'} \end{bmatrix}.$$

(3) N Unit Cells and Problem Geometry

In general, an analog HLN implementation has N unit cells, where each cell may or may not be coupled to other unit cells via a resistive bridge. One can extend the above analysis to the general case of N unit cells. This leads to a differential equation of the same form:

$$\dot{\mathbf{v}} = \mathbf{C}^{-1}[\mathbf{J}_{\mathrm{s}}\alpha - \mathbf{J}\mathbf{v} - \mathbf{I}_{\mathrm{NL}}], \tag{122}$$

where

$$\dot{\mathbf{v}} \equiv (\frac{dv_1}{dt}, \frac{dv_2}{dt}, ..., \frac{dv_N}{dt}), \ \mathbf{C} \equiv \mathrm{diag}(C_1, C_2, ..., C_N), \ \mathbf{J}_{\mathrm{s}} \equiv \mathrm{diag}(\frac{1}{R_1}, \frac{1}{R_2}, ..., \frac{1}{R_N}),$$
$$\mathbf{v} \equiv (v_1, v_2, ..., v_N), \ \alpha \equiv (\alpha_1, \alpha_2, ..., \alpha_N), \text{ and } \mathbf{I}_{\mathrm{NL}} \equiv (I_{\mathrm{NL},1}, I_{\mathrm{NL},2}, ..., I_{\mathrm{NL},N}).$$

In addition, the matrix $\mathbf{J}$ has elements given by

$$\mathbf{J}_{jj'} = \begin{cases} \frac{1}{R_j} + \sum_{\{k:\mathcal{A}_{jk}=1\}} \frac{1}{R_{jk}} & \text{if } j = j' \\ -\frac{1}{R_{jj'}} & \text{if } j \neq j' \end{cases} \tag{123}$$

Here, $\mathcal{A}$ is an adjacency matrix that represents the problem geometry. For example, some problems have one-dimensional geometry (e.g., time series data) and other problems have two-dimensional geometry (e.g., images). If $\mathcal{A}_{jk} = 1$, then cells j and k are coupled, while $\mathcal{A}_{jk} = 0$ when the pair is uncoupled. The matrices $\mathcal{A}$ and $\mathbf{J}$ allow the problem-specific geometry to be built into the circuit. This can improve performance of the model, since accounting for problem geometry leads to an inductive bias for the model, which often improves the trainability and generalization of the model. In practice, switches (e.g., voltage-gated transistors) can upload the problem geometry onto the circuit connectivity, as illustrated in Figure 18.

(4) Intuition

The differential equations in Eq. (122) are similar to the equation for a neural network in part because neural networks typically involve alternating layers of affine transformations followed by non-linear transformations. Taking the limit where the layers are infinitesimally small, it appears as if the affine transformation and non-linear transformation happen simultaneously. This limit of infinitesimally small layers is the limit associated with neural ODEs, and hence is the limit that our HLN corresponds to. Therefore, the right-hand-side of Eq. (122) involves affine and non-linear transformations acting at the same time. Namely the terms $\mathbf{J}\mathbf{v} - \mathbf{J}_{\mathrm{s}}\alpha$ provide an affine transformation on the vector $\mathbf{v}$ and the term $\mathbf{I}_{\mathrm{NL}}$ is a non-linear transformation on v.

Regarding the latter point, note that the jth non-linear element (NLE) is in parallel with the capacitor in the jth unit cell, and hence the voltage across the jth NLE is equal to v_j. Suppose the current-voltage characteristic for the jth NLE is given by the function f_j. Then we can write:

$$I_{\mathrm{NL},j} = f_j(v_j) \tag{124}$$

and

$$\mathbf{I}_{\mathrm{NL}} = (f_1(v_1), f_2(v_2), ..., f_N(v_N)). \tag{125}$$

By definition, f_j is a non-linear function, since it corresponds to a current-voltage characteristic for a NLE. Therefore, Eq. (125) corresponds to a non-linear transformation of the elements of the vector $\mathbf{v}$.

25.4 Physical implementations of non-linear elements

There are several different ways to implement non-linear elements (NLEs) for circuits, including diodes and transistors. The choice of NLE affects the kind of activation function that gets implemented in the neural ODE.

Consider diodes as NLEs. The current-voltage characteristic for a diode, whether under forward or reverse bias, has positive feedback. This means that the more voltage applied to the diode, the higher the conductivity. Since the NLE is in parallel with the capacitor in the unit cell, this leads to a negative feedback on the voltage vector $\mathbf{v}$, which can be seen from the negative sign appearing on the non-linear term in Eq. (122). Since the diode's current-voltage characteristic has a very sharp rise with applied voltage, this leads to a threshold effect where the non-linear drift term in Eq. (122) becomes very large as $\mathbf{v}$ increases. This leads to to a sharp negative feedback on $\mathbf{v}$, such that the magnitude of $\mathbf{v}$ saturates at some value. This is similar to certain activation functions employed in neural networks. Namely, there are some activation functions that have a saturation effect, such as the sigmoid function and the hyperbolic tangent function. In this sense, choosing the NLE to be a diode mimics these kinds of activation functions.

Alternatively, consider using a transistor as the NLE. Indeed, the current-voltage characteristic for a transistor can be non-linear, with the form of the function depending on whether one is looking at the input or output characteristic. For the transistor's common emitter and common base configurations, the input characteristic is convex with the current rising sharply with the applied voltage. In this case (i.e., when considering the input characteristic), the situation is similar to the case of the diode, and the corresponding activation function shows a saturation effect, like the sigmoid function and the hyperbolic tangent functions.

On the other hand, when employing a transistor and using the output characteristic, the current-voltage characteristic is concave and exhibits saturation. In turn, this leads to a convex activation function with positive feedback. Activation functions that have this positive feedback include the softplus function and the rectified linear unit (ReLU) function. The ReLU in particular is commonly employed in neural networks. Hence using the output characteristic of a transistor is a strategy for obtaining activation functions that have positive feedback.

25.5 Uploading Data to the Analog HLN

Each unit cell corresponds to a feature in the data. At time $t = 0$, a feature vector $\mathbf{x}$ is encoded into the hidden layer value $\mathbf{h}_0$. (For example, $\mathbf{x}$ could be a feature vector $\mathbf{x}^{(m)}$ from the training dataset.) This encoding may involve an digital-to-analog converter, which converts a digital signal associated with $\mathbf{x}$ into an analog signal. Once in analog form, $\mathbf{x}$ can be stored on the analog HLN by appropriately charging the capacitors in the HLN. Here, the jth feature of $\mathbf{x}$ is mapped to the capacitor in the jth unit cell, possibly with some permutation of the indices. Specifically, the initial voltage across the jth capacitor is set to be $v_j = x_j$ at time $t = 0$.

The notation $\mathbf{v}$ represents the physical vector of voltages across the capacitors in the unit cells and also corresponds mathematically to the vector $\mathbf{h}$ associated with the hidden layer values. Hence $\mathbf{v}$ and $\mathbf{h}$ are essentially interchangeable from a notation perspective.

25.6 Analog Augmented HLN

A feature vector $\mathbf{x}$ can be encoded into the initial hidden layer value $\mathbf{h}_0$. However, more generally, $\mathbf{x}$ can be encoded into a larger dimensional space, by padding the vector with zeros. This concept leads to what is known as augmented neural ODE. Augmented neural networks aim to expand the dimensionality of the feature space in order to more easily separate the data. Neural networks (without augmentation) may not be able to implement functions where trajectories intersect and hence neural ODEs may not be able to represent all functions. Augmented neural networks represented by augmented neural ODEs are useful for addressing this issue.

Figure 34 illustrates an analog version of an augmented neural network that executes an augmented neural ODE. The concept illustrated in Figure 34 can be used as the basis for constructing the HLN, which can be thought of as an augmented HLN. Mathematically, the initial hidden layer values can be written as a direct sum of the datapoint $\mathbf{x}$ and a vector of zeros, as follows:

$$h(0) = x(0) \bigoplus \mathbf{0}. \tag{126}$$

Physically this involves a device with more than N unit cells. Namely, the device would have $N + N_a$ unit cells where N_a is the dimension of the zero vector in Eq. (126).

The augmented HLN could have some non-trivial connectivity between the N unit cells in data space and the N_a unit cells in the additional space. This is illustrated in Figure 34. From the initial time to some later time t, the additional features can evolve to a non-trivial state $a(t)$, and the overall state can be written as:

$$h(t) = x(t) \bigoplus a(t). \tag{127}$$

The rest of the protocol can proceed similarly to the case without augmentation.

25.7 Making predictions from the final hidden layer

The final hidden layer value (i.e., the value at $t = 1$) is denoted as h_1.

In the context of supervised machine learning, the goal is to convert h_1 into a prediction for the output y. Suppose that there is a map, which could be a probabilistic map, that maps h_1 to y. This map can be described via the conditional distribution $p(y|h_1)$, where

$$p(y|h_1) = p(y|x, w) \tag{128}$$

since specifying the h_1 value is equivalent to specifying the input x and the weights w. The quantity appearing in Eq. (128) is the same quantity appearing in Eq. (108).

Assume that there may be a process at the end of the HLN whereby the last hidden layer value h_1 is converted to an output y. Here, the subsystem or device associated with this process is called the Output Predictor (OP). The OP takes h_1 as its input and outputs a y value, via either a deterministic or probabilistic function.

Physically speaking, the OP could be implemented with an analog device or a digital device. If h_1 has an analog form and y has a digital form, then the OP can use analog-to-digital converters and digital-to-analog converters to convert between the different types of signals. Alternatively, it is possible for h_1 and y to either both be analog or both be digital, in which case no conversion (between different signal types) is necessary.

26 Weight Diffuser

26.1 Overview of Weight Diffuser

Diffusion is a physical thermodynamic process that is often mimicked mathematically in machine learning models. This raises the question of whether the diffusion in these mathematical models could be performed with a physical thermodynamic device.

In the context of Bayesian neural networks, the weights can be modeled as undergoing a diffusion process, which adds uncertainty to the weights. For this application, a physical thermodynamic device called a Weight Diffuser (WD) devices diffuses the weights over time. The thermodynamic aspect of the WD device is supplied by an analog stochastic noise source, which leads to physical diffusion of the system's state variable.

In what follows, we first describe a WD device for the prior distribution. Then we discuss a WD device for the posterior distribution. Finally, we discuss how the two devices can be represented by a single device that is equipped with switches, to toggle between the prior and posterior cases.

26.2 Prior Weight Diffuser

The prior distribution is typically assumed to be relatively simple, and consequently the WD device for the prior distribution is also relatively simple. In some cases, the prior weight diffuser may not be necessary, since the simplified loss function in Eq. (115) does not involve the prior distribution. On the other hand, the loss function in Eq. (110) does involve the prior distribution, and hence the prior weight diffuser would be useful in this case. In this sense, the prior weight diffuser is an optional component in the overall system for Bayesian deep learning.

One way to construct the weight diffuser is by building one unit cell for each weight. Suppose that there are W weights in total. Typically, there are more weights than neurons, $W > N$. In any case there is no reason to assume that $W = N$, so in general the number of unit cells in the WD can be different than the number of unit cells in the HLN.

(1) The Unit Cell of a Prior Weight Diffuser

Figure 35 provides a circuit diagram for the unit cell for the prior weight diffuser. The unit cell includes a capactor C_i, a resistor R_i, and a stochastic noise source B_i, all of which are in series.

The voltage across the capacitor in the unit cell is denoted as $\tilde{v}_i$. The abstract weight w_i is encoded in the physical voltage $\tilde{v}_i$, which means that the dynamical state variable for the WD device is:

$$\tilde{v}_i = w_i \tag{129}$$

Because there is a stochastic noise source in the circuit, the state variable evolves according to a stochastic differential equation (SDE). For the unit cell in Figure 35, the SDE for the state variable is:

$$-R_i C_i \mathrm{d}\tilde{v}_i = \tilde{v}_i \mathrm{d}t + \sigma_i \mathrm{d}B_i \tag{130}$$

where σ_i is the standard deviation associated with the stochastic noise source.

(2) Two Coupled Cells

Now consider two coupled unit cells with either resistive coupling or capacitive coupling. Figure 35 shows the case of capacitive coupling between two unit cells. For capacitive coupling, the SDE associated with two unit cells is:

$$-\mathbf{RC}\, d\tilde{\mathbf{v}} = \tilde{\mathbf{v}} dt + \sigma d\mathbf{B}, \tag{131}$$

where

$$\mathbf{C} \equiv \begin{bmatrix} C_i + C_{ii'} & C_{ii'} \\ C_{ii'} & C_{i'} + C_{ii'} \end{bmatrix}, \ \mathbf{R} \equiv \begin{bmatrix} R_i & 0 \\ 0 & R_{i'} \end{bmatrix}, \ \sigma \equiv \begin{bmatrix} \sigma_i & 0 \\ 0 & \sigma_{i'} \end{bmatrix}, \ d\mathbf{B} \equiv \begin{bmatrix} dB_i \\ dB_{i'} \end{bmatrix}, \ \tilde{\mathbf{v}} \equiv \begin{bmatrix} \tilde{v}_i \\ \tilde{v}_{i'} \end{bmatrix}.$$

(3) W Unit Cells and problem geometry

The extension to W unit cells is straightforward. Each cell may or may not be coupled to other cells, where the choice of whether or not to couple two cells can be based on problem geometry. (This is similar to the discussion above on the HLN.) Figure 10 (discussed above) illustrates how to use the adjacency matrix $\mathcal{A}$ to determine the connectivity of the unit cells.

Assuming that the coupling between cells involves a capacitive bridge, the SDE for the W unit cell case is:

$$d\tilde{\mathbf{v}} = \mathbf{C}^{-1}\mathbf{R}^{-1}\left[-\tilde{\mathbf{v}}dt + \sigma d\mathbf{B}\right], \tag{132}$$

where the minus sign has been dropped from the stochastic term since it has no effect on the Brownian motion, which is Gaussian distributed with zero mean. The various objects are defined as:

$$\mathbf{R} \equiv \text{diag}(R_1, R_2, ..., R_W), \ \sigma \equiv \text{diag}(\sigma_1, \sigma_2, ..., \sigma_W), \ \tilde{\mathbf{v}} \equiv (\tilde{v}_1, \tilde{v}_2, ..., \tilde{v}_W), \ d\mathbf{B} \equiv (dB_1, dB_2, ..., dB_W).$$

In addition, the matrix $\mathbf{C}$ has elements given by

$$\mathbf{C}_{ii'} = \begin{cases} C_i + \sum_{\{k:\mathcal{A}_{ik}=1\}} C_{ik} & \text{if } i = i' \\ C_{ii'} & \text{if } i \neq i' \end{cases} \tag{133}$$

where the adjacency matrix $\mathcal{A}$ appears here again. In principle, the adjacency matrix for the WD can be different than the adjacency matrix for the HLN. For simplicity, we present the case where they are the same, although they could be different.

26.3 Posterior Weight Diffuser

The architecture for the Posterior Weight Diffuser is similar to that of the Prior Weight Diffuser.

Once again, consider W unit cells. The unit cell for the posterior weight diffuser is illustrated in Figs. 37A and 37B. This includes the case where the posterior drift network (PDN) is analog (Fig. 37A), and also includes the case where the PDN is digital (Fig. 37B). In the digital PDN, an analog-to-digital converter converts the measured voltage on the capacitor to a digital signal, and a digital-to-analog converter converts the output of the PDN to an analog voltage that is applied inside the unit cell. The analog PDN operates without converters. Other than this feedback loop involving the PDN, the unit cell is similar to that of the prior weight diffuser.

Once again, capacitive bridges can connect the unit cells, and the connectivity can be chosen according to an adjacency matrix $\mathcal{A}$ and hence according to the problem geometry. Again, this is illustrated by Figure 10. It is reasonable to choose the same connectivity for the prior weight diffuser and the posterior weight diffuser.

Assuming capacitive coupling between the unit cells, the overall SDE for a state vector associated with the W unit cells is given by:

$$d\tilde{\mathbf{v}} = \mathbf{C}^{-1}\mathbf{R}^{-1}\left[-\tilde{\mathbf{v}}dt + \mathbf{s}^{(\phi)}(\tilde{\mathbf{v}}, t)dt + \sigma d\mathbf{B}\right]. \tag{134}$$

A difference here, relative to the prior weight diffuser, is the addition of the drift term $\mathbf{s}^{(\phi)}(\tilde{\mathbf{v}}, t)$ which

can be a complicated function of the time t and the state vector $\tilde{\mathbf{v}}$. The PDN is discussed in more detail below.

The Posterior Weight Diffuser is parameterized with parameters ϕ that appear in the drift term. In addition, the initial condition is also has parameters. The initial condition is:

$$\tilde{\mathbf{v}}^{(\theta)}(0) = \mathbf{w}^{(\theta)}(0) \tag{135}$$

Here, θ represents the parameters, and $\tilde{\mathbf{v}}^{(\theta)}(0)$ and $\mathbf{w}^{(\theta)}(0)$ are, respectively, the physical version and abstract version of the initial condition. Then the Posterior Weight Diffuser can be written as an integral equation:

$$\tilde{\mathbf{v}}(t) = \tilde{\mathbf{v}}^{(\theta)}(0) + \int_{t'=0}^{t'=t} \mathbf{C}^{-1}\mathbf{R}^{-1}\left[-\tilde{\mathbf{v}}dt' + \mathbf{s}^{(\phi)}(\tilde{\mathbf{v}}, t)dt' + \sigma d\mathbf{B}\right]. \tag{136}$$

Here, the set of parameters is

$$\psi = \{\theta, \phi\} \tag{137}$$

During the training process, a digital processor stores the values of the ψ parameters and propose updates to these parameters during an optimization routine. This is discussed more below.

The initial condition $\tilde{\mathbf{v}}^{(\theta)}(0)$ can be set by the digital processor. This involves the processor outputting a digital signal that is converted to analog by a digital-to-analog converter. Then this analog signal is used to appropriately charge the capacitors in each unit cell of the Posterior Weight Diffuser (i.e., the capacitors illustrated in Fig. 37). This can be thought of as uploading the initial condition onto the analog device for the Posterior Weight Diffuser.

26.4 Prior and posterior weight diffusers as the same device

Figure 40 shows a Weight Diffuser device with switches, such as voltage-gated transistors, that can be toggled to swith the Weight Diffuser between a Prior and Posterior Weight Diffuser settings. This allows a single device to perform the functions of both the Prior and Posterior Weight Diffusers.

26.5 Sources of Brownian motion

Thus far we have described the stochastic noise source in the Weight Diffuser as an abstract device. This abstract circuit element is illustrated as B_i in Figures 35 and 37. From a mathematical perspective, it is desirable for the stochastic noise source to be uncorrelated in time and have a Gaussian distribution with zero mean.

The stochastic noise source can be a thermal noise source, a shot noise source, or a source of both thermal noise and shot noise. A variable amplifier can amplify the output of the stochastic noise source, making it possible to tune the standard deviation σ appearing in the differential equation (e.g., as in Eqs. (133) and (134)).

One type of possible stochastic noise is thermal noise. Thermal noise, also called Johnson-Nyquist noise, comes from the random thermal agitation of the charge carriers in a conductor, resulting in fluctuations in voltage or current inside the conductor. Thermal noise is Gaussian and has a flat frequency spectrum (white noise) with fluctuations in the voltage of standard deviation

$$v_{tn} = \sqrt{4k_B T R \Delta f}, \tag{138}$$

where R is the resistance of the conductor, k_B is the Boltzmann constant, T is the absolute temperature and Δf is the frequency bandwidth. The amplitude of the voltage fluctuations can be controlled by

changing the temperature or the resistance. In practice, a thermal noise source can be implemented using a large resistor in series with a voltage amplifier. Alternatively, a thermal noise source can be implemented using a tunable resistor (e.g., a transistor operated in the linear regime) where the noise amplitude scales with the value of the resistance.

Another type of stochastic noise is shot noise. Shot noise arises from the discrete nature of charge carriers and from the fact that the probability of a charge carrier crossing a point in a conductor at any time is random. This effect is particularly important in semiconductor junctions where the charge carriers should overcome a potential barrier to conduct a current. The probability of a charge carrier passing over the potential barrier is an independent random event. This induces fluctuations in the current through the junction. Shot noise is Gaussian and has a flat frequency spectrum (white noise) with fluctuations in the current of standard deviation

$$I_{sn} = \sqrt{2q|I|\Delta f}, \tag{139}$$

where I is the current through the junction, $\mathbf{q}$ is the electron charge and Δf is the frequency bandwidth. The amplitude of the current fluctuations can be controlled by changing the magnitude of the DC current passing through the junction. In practice, a source of shot noise can be implemented using a pn diode (for example, a Zener diode in reverse bias configuration) in series with a controllable current source.

27 Interfacing the HLN with the WD

27.1 Outputting weights from the weight diffuser

Figure 38 shows how to output weights from the weight diffuser devices to the HLN. The voltages across the capacitors in the unit cells of the WD are picked off to give a voltage vector $\tilde{\mathbf{v}}(t) = \mathbf{w}(t)$.

The elements of this voltage vector can be permuted with a permutation map. Physically speaking, this permutation map corresponds to a routing scheme for the wires associated with the $\mathbf{w}(t)$ vector. Routing the wires in different ways yields different permutations of the elements in this vector. This yields a new vector:

$$\omega(t) = P(\mathbf{w}(t)). \tag{140}$$

Here, $P(\cdot)$ is a permutation function. The vector $\omega(t)$ can then be directly outputted to the HLN device.

The permutation provides flexibility in how to assign the weight values to specific parameters appearing in the HLN device.

27.2 Inputting weights into the HLN

Figure 39 illustrates how to input weights from the WD to the HLN.

The first step is to parameterize the HLN in terms of voltages. The HLN involves variable resistors that can be implemented by voltage-controlled circuit elements. For example, Figure 39 shows the variable resistors as voltage-gated transistors, where the gate voltage controls the value of the resistance.

Specifically, the variable resistances are:

$$R_j = f_j(\beta_j), \qquad R_{jj'} = f_{jj'}(\Gamma_{jj'}). \tag{141}$$

Here, β_j is an element of a voltage vector $\beta = \{\beta_j\}$, and $\Gamma_{jj'}$ is an element of a voltage matrix $\mathbf{\Gamma} = [\Gamma_{jj'}]$. The functions f_j and $f_{jj'}$ are device-dependent functions that translate the gate voltage into a resistance.

Taken together, the voltage vectors α and β and the voltage matrix $\mathbf{\Gamma}$ represent the set of parameters

used by the HLN device. We can write this set as:

$$\omega(t) = \{\alpha(t), \beta(t), \mathbf{\Gamma}(t)\} \tag{142}$$

where these parameters depend on time t. The notation $\omega(t)$ for this set of parameters corresponds precisely to the output vector of the WD device, given in Eq. (140).

We applied a permutation function in Eq. (140) to account for a routing scheme that maps the outputs of the WD device to the inputs of the HLN device.

28 Posterior drift network

28.1 Overview of PDN

As discussed previously and illustrated in Fig. 37, the Posterior Drift Network (PDN) plays a central role in generating the posterior distribution for the weights. The PDN takes in real-time measurements of the the weights and outputs drift values as voltages to be applied in the WD unit cells. In what follows, we discuss several different approaches to constructing the PDN. We first consider a digital neural network for the PDN. Then we consider the PDN as an analog neural network with digitally stored time-dependent weights. Finally we consider the PDN as an analog device with analog time evolution.

28.2 PDN as a digital neural network

One approach is to use a digital device to store and process the PDN. This digital device could be a CPU or an FPGA. A benefit of using an FPGA is that it can have low latency when interacting with an analog circuit. Hence the feedback loop between the FPGA and analog circuit in the WD could occur quickly.

Figure 41 shows how the PDN interacts with the unit cells of the WD, in the case that the PDN is stored on a digital device. The PDN takes in the entire weight vector as input to the neural network. Analog-to-digital and digital-to-analog converters are used to appropriately convert the signals between the analog and digital domains.

Relative to an analog PDN, a digital PDN has the advantage of being flexible in its design, since the construction can be modified in software (rather than in hardware). In addition, digital neural networks often allow for more parameters and hence more expressibility than their analog counterparts.

On the other hand, one drawback of a digital PDN is latency. The analog-to-digital and digital-to-analog converters can add latency to the feedback loop between the WD and the PDN. In addition, the forward pass through the neural network involves matrix-vector multiplications and hence can take time, adding latency.

28.3 PDN as an analog neural network with time-dependent parameters

Figure 42 shows a possible architecture for an analog PDN. This analog neural network has alternating layers of affine transformation followed by non-linear transformations.

The non-linear transformation is illustrated in Figure 43. This can involve a non-linear element (NLE) that is in series with a resistor. For example, the NLE could correspond to a diode or a transistor, and Sec. 25.4 discusses these NLEs in more detail. The resistor in series with the NLE makes it possible to read off the current through the NLE as an output voltage. Hence the output voltage is a non-linear

function of the input voltage, which is the desired feature of activation functions that are commonly used in neural networks.

Figure 43 elaborates on possible affine transformations. This can involve a layer of resistors in series with the inputs, followed by a wire combining the outputs. This functions to produce an output voltage that is a weighted average of the input voltages. This produces a particular affine transformation. There can also be an amplifier after each layer in order to give more flexibility to this affine transformation (e.g., to allow it to change the norm of the voltage vector).

The resistances in the affine layer are free parameters that get trained during the optimization process. Moreover, these resistances can be time dependent. Each resistance can be expressed as a time-dependent function that is parameterized in some way. For example, the resistance can be expressed as a linear function of time, in which case the slope and intercept on that linear function would be trainable parameters. Physically speaking, the resistors can be transistors, the time-dependent resistances could be implemented with time-dependent voltages applied to the gates of the transistors.

Overall, this PDN construction has the benefit of essentially no latency. This is because ADCs and DACs are not needed in the interface between the WD and the PDN. It is also because a forward pass through the PDN is essentially instantaneous, since the voltage propagates through the neural network at the speed of light. On the other hand, this construction may have less flexibility to alter the design and possibly less expressibility than a digital PDN.

28.4 PDN as an analog device with analog time evolution

An alternative way to implement an analog PDN is as follows. Instead of using time-dependent parameters, time dependence can be incorporated via a differential equation that describes the time evolution of the PDN.

Let $\mathbf{s}$ be the state variable associated with the PDN. This variable represents the drift vector that the PDN outputs and feeds to the WD. With a high degree of generality, we can write a differential equation for the time evolution of $\mathbf{s}$ as follows:

$$\frac{d\mathbf{s}}{dt} = \mathbf{e}^{(\phi)}\left(t, \mathbf{s}, \mathbf{w}, \frac{d\mathbf{w}}{dt}\right) \tag{143}$$

where $\mathbf{e}^{(\phi)}$ is some parameterized function, with ϕ being the parameters, and, for the PDN, $\mathbf{s}^{(\phi)}(0)$ is a chosen initial condition. This initial condition can also be viewed as being parameterized.

With this general framework, we can write the differential equation for the overall system, involving the HLN, WD, and PDN. The state variables of these three devices evolve over time according to the following system of differential equations:

$$d\begin{bmatrix}\mathbf{h}_t \\ \mathbf{w}_t \\ \mathbf{s}_t\end{bmatrix} = \begin{bmatrix}\mathbf{f}_h(t, h_t, w_t) \\ \mathbf{s}_t + \mathbf{w}_t \\ \mathbf{e}^{(\phi)}\left(t, \mathbf{s}, \mathbf{w}, \frac{d\mathbf{w}}{dt}\right)\end{bmatrix} dt + \begin{bmatrix}\mathbf{0} \\ g_w \\ \mathbf{0}\end{bmatrix} dB_t \tag{144}$$

This approach has the elegance of placing all three devices on the same footing. Moreover, all three devices could be physical systems that evolve simultaneously with each other over time.

Next, consider a possible form for the function $\mathbf{e}^{(\phi)}$. $\mathbf{s}$ can be a function of both time t and the weight vector $\mathbf{w}$, which is also a function of time, $\mathbf{w} = \mathbf{w}(t)$. After all, $\mathbf{w}(t)$ evolves according to the differential equation in Eq. (134). In summary, $\mathbf{s}$ depends on time through two sources: it depends directly on t and it depends indirectly on t via $\mathbf{w}(t)$.

This motivates an approach based on the total derivative. The total derivative formula can be written

as follows:

$$\frac{d\mathbf{s}}{dt} = \frac{\partial \mathbf{s}}{\partial t} + \sum_{i=1}^{W} \frac{\partial \mathbf{s}}{\partial w_i} \frac{dw_i}{dt} = \frac{\partial \mathbf{s}}{\partial t} + \nabla_{\mathbf{w}} \mathbf{s} \cdot \frac{d\mathbf{w}}{dt} \tag{145}$$

Here, $\nabla_{\mathbf{w}}$ denotes the gradient with respect to $\mathbf{w}$. The total derivative formula captures the dependence on both t and $\mathbf{w}$. Moreover, it provides a recipe for how to design an analog PDN.

Suppose that there are functions that can approximate the partial derivatives in Equation (145). These functions can be used to construct an analog PDN. Define the functions $\mathbf{q}^{(\phi)}(t, \mathbf{w})$ and $\mathbf{r}^{(\phi)}(t, \mathbf{w})$. Rewriting Equation (145) yields a model for the drift function:

$$\frac{d\mathbf{s}}{dt} = \mathbf{q}^{(\phi)}(t, \mathbf{w}) + \mathbf{r}^{(\phi)}(t, \mathbf{w}) \cdot \frac{d\mathbf{w}}{dt} \tag{146}$$

with

$$\mathbf{q}^{(\phi)}(t, \mathbf{w}) \approx \frac{\partial \mathbf{s}}{\partial t} \quad \text{and} \quad \mathbf{r}^{(\phi)}(t, \mathbf{w}) \approx \nabla_{\mathbf{w}} \mathbf{s}. \tag{147}$$

In other words, the trainable function $\mathbf{q}^{(\phi)}(t, \mathbf{w})$ provides a model for the partial derivative $\frac{\partial \mathbf{s}}{\partial t}$, and the trainable function $\mathbf{r}^{(\phi)}(t, \mathbf{w})$ provides a model for the gradient $\nabla_{\mathbf{w}} \mathbf{s}$. This constitutes a splitting of the $\mathbf{e}^{(\phi)}$ function into two functions $\mathbf{q}^{(\phi)}$ and $\mathbf{r}^{(\phi)}$ that describe different contributions to the time derivative of the drift function.

Figure 44 provides a schematic diagram of a circuit 4440 that can output drift values from the PDN, using the total derivative approach. This circuit has several components:

1. voltage sources 4441 whose outputs are $\mathbf{r}^{(\phi)}(t, \mathbf{w})$;
2. voltage mixers 4442 that multiply the components of $\mathbf{r}^{(\phi)}(t, \mathbf{w})$ with the corresponding components of $\frac{d\mathbf{w}}{dt}$, where the mathematical terms $\frac{d\mathbf{w}}{dt}$ are encoded in voltage vectors;
3. a voltage source 4443 whose output is $\mathbf{q}^{(\phi)}(t, \mathbf{w})$;
4. voltage adders 4444 that add the voltages from the voltage sources 4441 and 4443 together to produce overall voltage signals $\mathbf{e}^{\phi}(t, \mathbf{w}, \frac{d\mathbf{w}}{dt}) = \mathbf{q}^{(\phi)}(t, \mathbf{w}) + \mathbf{r}^{(\phi)}(t, \mathbf{w}) \cdot \frac{d\mathbf{w}}{dt}$.
5. integrator circuits 4445 that output voltages $\mathbf{i}^{(\phi)}(t')$ that are the time integrals of the signals $\mathbf{e}^{\phi}(t, \mathbf{w}, \frac{d\mathbf{w}}{dt})$, of the form $\mathbf{i}^{(\phi)}(t') = \int_{t=0}^{t=t'} \mathbf{e}^{\phi}(t, \mathbf{w}, \frac{d\mathbf{w}}{dt}) dt$; and
6. voltage adders 4447 that add the output voltages of the integrators, $\mathbf{i}^{(\phi)}(t')$, with the initial condition $\mathbf{s}^{(\phi)}(0)$ from a voltage source 4446, to produce an output voltage 4447 corresponding to the predicted score value $\mathbf{s}^{(\phi)}(t') = \mathbf{s}^{(\phi)}(0) + \mathbf{i}^{(\phi)}(t')$.

Figure 44 depicts several circuits as black boxes. This includes the adder circuits 4444 and 4447, the integrator circuits 4445, and the circuits 4441 and 4443 associated with the voltage sources $\mathbf{q}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$ and $\mathbf{r}_{\boldsymbol{\theta}}(\mathbf{v}, \tau)$. Figure 17 shows examples of an adder circuit (left) and an integrator circuit (right) suitable for use in the score device 340.

The voltage vector $\frac{d\mathbf{w}}{dt}$ input to each voltage mixer can be obtained as follows. In the Weight Diffuser, the unit cell has a resistor that is in series with the capacitor, as shown in Figure 37. The voltage across this resistor is proportional to the time derivative of the capacitor's voltage, and hence is proportional to $\frac{dw_i}{dt}$. Therefore, picking off the voltages across the resistors in all of the unit cells of the WD yields the voltage vector $\frac{d\mathbf{w}}{dt}$. This voltage vector can then be fed into the PDN's voltage mixer.

A circuit can provide the voltage sources $\mathbf{q}^{(\phi)}(t, \mathbf{w})$ and $\mathbf{r}^{(\phi)}(t, \mathbf{w})$. One choice is to build these sources as layered neural networks, with alternating layers of affine transformations and non-linear transformations. Hence, $\mathbf{q}^{(\phi)}(t, \mathbf{w})$ and $\mathbf{r}^{(\phi)}(t, \mathbf{w})$ can be constructed based on the circuit structure shown in Fig. 42, and the affine layer can be constructed in a manner similar to that shown in Fig. 43.

28.5 Incorporating problem geometry into the PDN

There are several different approaches to constructing the PDN. These approaches can benefit from accounting for problem geometry.

As an example, when using an analog PDN, the problem geometry can be incorporated into the circuit structures shown in Figures 42 and 43. Each affine transformation in Figures 42 and 43 is mathematically represented by some matrix A. The problem geometry can be incorporated into each of the A matrices associated with these affine transformations. For example, these A matrices can be sparse, with non-zero elements only corresponding to the connectivity of the problem. In other words, one can choose these A matrices to have a similar structure to the adjacency matrix $\mathcal{A}$ that has been discussed previously. Matching the zero elements of the A matrices to the zero elements of the adjacency matrix $\mathcal{A}$ provides a recipe for incorporating problem geometry into the analog PDN.

Physically speaking, this corresponds to limiting the number of wires uses in the circuit structure in Fig. 43. In other words, the wires in Fig. 43 can be bundled together if the adjacency matrix $\mathcal{A}$ has non-zero elements associated with those wires. Hence, the adjacency matrix can be used to guide incorporation of problem geometry in the construction of the circuit in Fig. 43.

29 Loss Evaluator

29.1 Estimators for the ELBO loss function

Recall that the loss function can be written as follows:

$$\mathcal{L}_{ELBO} = \mathbb{E}_{q(w)}\left[\log p(D|w) - \frac{1}{2}\int_0^1 \|u(t,w_t,\phi)\|_2^2 dt\right] \tag{148}$$

where the function u is given by

$$u(t,w_t,\phi) = g_w(t,w_t)^{-1}[f_w^p(t,w_t) - f_w^q(t,w_t,\phi)] \tag{149}$$

The loss terms are denoted κ and λ. We can write $\mathcal{L}_{ELBO} = \kappa - \lambda$, with

$$\kappa = \mathbb{E}_{q(w)}\left[\log p(D|w)\right], \qquad \lambda = \mathbb{E}_{q(w)}\left[\frac{1}{2}\int_0^1 \|u(t,w_t,\phi)\|_2^2 dt\right]. \tag{150}$$

An unbiased estimator can compute the loss function (or its gradient). This estimator can be low precision, which conserves resources.

Next, consider some estimators for the loss function. Consider sampling from $q(w)$. Suppose we draw K samples, $w^{(k)} \sim q(w)$, with $k = 1, ..., K$. Then we can define an unbiased estimator of the loss function via:

$$\widehat{\mathcal{L}}_1 = \frac{1}{K}\sum_{k=1}^{K}\left[\log p(D|w^{(k)}) - \frac{1}{2}\int_0^1 \|u(t,w_t^{(k)},\phi)\|_2^2 dt\right] = \widehat{\kappa}_1 - \widehat{\lambda}_1 \tag{151}$$

where

$$\widehat{\kappa}_1 = \frac{1}{K}\sum_{k=1}^{K}\log p(D|w^{(k)}), \qquad \widehat{\lambda}_1 = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{2}\int_0^1 \|u(t,w_t^{(k)},\phi)\|_2^2 dt \tag{152}$$

One suitable unbiased estimator for the first term, $\log p(D|w^{(k)})$, is known as the log likelihood.

Consider a dataset $D = \{x^{(m)}, y^{(m)}\}_{m=1}^{M}$ with M datapoints. The log likelihood can be written as

$$\zeta^{(k)} \equiv \log p(D|w^{(k)}) = \sum_{m=1}^{M} \log p^{(m,k)} = \sum_{m=1}^{M} \zeta^{(m,k)} \tag{153}$$

where $\zeta^{(m,k)} = \log p^{(m,k)}$, and $p^{(m,k)} = \Pr(y^{(m)}|x^{(m)}, w^{(k)})$ is the probability that the model outputs $y^{(m)}$ when the input is $x^{(m)}$ and the weights are $w^{(k)}$.

Let $y_\ell^{(m,k)}$ be a random variable associated with the output of the ℓth trial of running the model for an input $x^{(m)}$. We can define a binary random variable $z_\ell^{(m,k)} \in \{0, 1\}$ such that $z_\ell^{(m,k)} = 1$ if $y_\ell^{(m,k)} = y^{(m)}$, and $z_\ell^{(m,k)} = 0$ if $y_\ell^{(m,k)} \neq y^{(m)}$.

An unbiased estimator for the log likelihood can be based on inverse binomial sampling (IBS). IBS corresponds to collecting samples $z_\ell^{(m,k)}$ until a sample returns a value of one, i.e., keep sampling until $z_\ell^{(m,k)} = 1$. Denote the number of samples drawn as $L^{(m,k)}$, where $L^{(m,k)}$ is a random variable. We introduce the estimator $\widehat{\zeta}^{(m,k)}$ where

$$\widehat{\zeta}^{(m,k)} = \begin{cases} 0, & \text{if } L^{(m,k)} = 1 \\ -\sum_{l=1}^{L^{(m,k)}-1} (\frac{1}{\ell}), & \text{if } L^{(m,k)} \geq 1 \end{cases} \tag{154}$$

This is an unbiased estimator, which means that $\mathbb{E}(\widehat{\zeta}^{(m,k)}) = \zeta^{(m,k)}$. Hence we can define another unbiased estimator for the ELBO loss function as follows:

$$\widehat{\mathcal{L}}_2 = \widehat{\kappa}_2 - \widehat{\lambda}_1 \tag{155}$$

where

$$\widehat{\kappa}_2 = \frac{1}{K} \sum_{k=1}^{K} \sum_{m=1}^{M} \widehat{\zeta}^{(m,k)} \tag{156}$$

Another suitable biased estimators is based on fixed sampling. Although it is biased, it may be easier to use analog hardware to estimate it, compared to $\widehat{\kappa}_2$. Suppose that there are $L^{(m,k)}$ samples for the kth weight trajectory and the mth datapoint from the dataset. Here, $L^{(m,k)}$ is a fixed value. The quantity

$$\widehat{\eta}^{(m,k)} = \log \left(\frac{\sum_{\ell=1}^{L^{(m,k)}} z_\ell^{(m,k)} + 1}{L^{(m,k)} + 1} \right) \tag{157}$$

$\widehat{\eta}^{(m,k)}$ is an estimator for $\zeta^{(m,k)}$, although it can have a bias that becomes more pronounced when $p^{(m,k)}$ is small. We then introduce a (biased) estimator for κ and $\mathcal{L}$ as follows:

$$\widehat{\kappa}_3 = \frac{1}{K} \sum_{k=1}^{K} \sum_{m=1}^{M} \widehat{\eta}^{(m,k)}, \qquad \widehat{\mathcal{L}}_3 = \widehat{\kappa}_3 - \widehat{\lambda}_1 \tag{158}$$

In addition, one could sample over datapoints in D, rather than include the entire sum over all datapoints in D, to define an additional estimator of the ELBO loss function. This is often known as a minibatch approach.

What follows covers how to separately estimate the two terms, κ and λ.

29.2 Estimating the log likelihood term, κ

The loss term κ can be estimated using two approaches: one based on the unbiased estimator $\widehat{\kappa}_2$, and one based on the biased estimator $\widehat{\kappa}_3$.

One way to estimate $\widehat{\kappa}_3$ is to use a single sample for each i and k. In other words, we can set $L^{(m,k)} = 1$. This choice is particularly well suited to the analog hardware, since each run of the WD device generates a new weight trajectory. Hence, to use more than one sample for each weight trajectory, the memory stores the weight trajectory. This memory storage could complicate the protocol and hence is beneficial to avoid. Choosing $L^{(m,k)} = 1$, we can compute:

$$\widehat{\kappa}_3 = \frac{1}{K} \sum_{m=1}^{M} \sum_{k=1}^{K} \log \left(\frac{z_1^{(m,k)} + 1}{2} \right) \tag{159}$$

This is done using the simultaneous time evolution of the HLN device and Posterior WD device. Start with a datapoint $x^{(m)}$ from the dataset D and uses it to initialize the input $\mathbf{h}(0)$ of the HLN device. The HLN device and the Posterior WD device then evolve over time simultanously, and this generates a particular weight trajectory $\mathbf{w}^{(k)}$. This also generates, at the output of the HLN, an output value $y_1^{(m,k)}$, which can be translated into a value for $z_1^{(m,k)}$ based on whether or not the output agrees with $y^{(m)}$. This $z_1^{(m,k)}$ value can be plugged into the expression in Eq. (159), and the process can be repeated for different elements $x^{(m)}$ from the dataset to compute $\widehat{\kappa}_3$.

The benefit of using $\widehat{\kappa}_3$ is that it is easy to compute with analog hardware, although it is a biased estimator. We can instead compute $\widehat{\kappa}_2$, which is unbiased, but it comes with possible computational difficulty.

Using the inverse binomial sampling (IBS) approach involves repeatedly running the HLN device with the same weight trajectory. This appears to require storing the weight trajectory in memory. Hence, $\widehat{\kappa}_2$ can involve the following protocol:

1. Choose a datapoint $x^{(m)}$ from the dataset D
2. Use this datapoint to initialize the input $\mathbf{h}(0)$ of the HLN device;
3. Evolve the HLN device and the Posterior WD device over time simulatenously, and this generates a particular weight trajectory $\mathbf{w}^{(k)}$ (this also generates, at the output of the HLN, an output value $y_1^{(m,k)}$, which can be translated into a value for $z_1^{(m,k)}$ based on whether or not the output agrees with $y^{(m)}$);
4. Store the weight trajectory $\mathbf{w}^{(k)}$in memory, e.g., in digital memory using an ADC;
5. Rerun the HLN device with the same datapoint $x^{(m)}$ and same weight trajectory $\mathbf{w}^{(k)}$ (this involves interfacing the memory device that stores the weight trajectory with the HLN, possibly using a DAC if the memory device is digital); and
6. Repeat the previous step until getting a value of $z_1^{(m,k)} = 1$, and the number of repetitions gives the value of $L^{(m,k)}$ to be used in the formula in Eq. (154).

This protocol for computing $\widehat{\kappa}_2$ is more complicated than the protocol for computing $\widehat{\kappa}_3$, since it involves a memory device to store the weight trajectories. Hence, there is a tradeoff between the complexity of the protocol and the bias of the estimator.

29.3 Digital time integration for the λ term

The λ loss term can be estimated using the following estimator:

$$\widehat{\lambda}_1 = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{2} \int_0^1 \|u(t, w_t^{(k)}, \phi)\|_2^2 dt, \tag{160}$$

which is an unbiased estimator for λ.

Figure 45 shows a system for computing $\widehat{\lambda}_1$. The weight diffuser 3310 exchanges weights for drift terms with the posterior drift network 3330. An amplifier 4510 coupled to the weight diffuser 3310 amplifies the weights and provides them to a componentwise adder 4520, which adds the amplified weights to drift terms from the posterior drift network 3330. An ADC 4530 digitizes the sums, which are in the analog domain, and provides them to a digital device 4540, such as an FPGA or CPU, for computing the time integral in Eq. (160).

Define the vector

$$\mathbf{d}(t) = -g_w(t, w_t)u(t, w_t, \phi) = f_w^q(t, w_t, \phi) - f_w^p(t, w_t) \tag{161}$$

This vector is produced as an analog voltage vector, using the outputs of the WD 3310 and the PDN 3330. For example, consider using the formulas $f_w^p(t, w_t) = -w_t$ and $f_w^q(t, w_t, \phi) = w_t + \mathbf{s}^{(\phi)}(t, w_t)$. With these formulas, we have

$$\mathbf{d}(t) = 2w_t + \mathbf{s}^{(\phi)}(t, w_t) \tag{162}$$

As shown in Figure 45, we can prepare this expression for $\mathbf{d}(t)$ by amplifying the output of the WD 3310 (by a factor of two) with the amplifier 4510 and then using the voltage adder 4520 to add each component of $2w_t$ and $\mathbf{s}^{(\phi)}(t, w_t)$, where the latter comes from the output of the PDN 3330. $\mathbf{d}(t)$ can be prepared as a voltage vector and passed through the ADC 4530, which feeds the resulting digital signal to the digital processor 4540. The digital processor 4540 multiplies $\mathbf{d}(t)$ by g^{-1}, then takes the norm squared of the resulting vector, then integrates over time, and finally sums over all samples (sum over k). The result of this computation is $\widehat{\lambda}_1$.

29.4 Analog time integration for the λ term

Alternative methods of computing $\widehat{\lambda}_1$ involve more computations on the analog device. After all, analog systems are efficient at computing time integrals. Hence we can compute the time integral in Eq. (160) on the analog device.

Figure 46 shows a system that uses analog time integration to compute $\widehat{\lambda}_1$. Like the system shown in Figure 45, the system in Figure 46 includes the amplifier 4510 and adder 4520 for generating the analog voltage vector $\mathbf{d}(t)$ as described above.

However, once $\mathbf{d}(t)$ is prepared, instead of passing it through an ADC, the system has analog circuitry (e.g., mixers 4610) that multiplies it by g^{-1}. Figure 46 shows how to do this multiplication for the special case where g^{-1} is a diagonal matrix, given by $g^{-1} = \mathrm{diag}(\gamma_1, \gamma_2, ..., \gamma_W)$. More generally, if g^{-1} is an arbitrary matrix, then an analog circuit can perform matrix-vector multiplication. The result of this multiplication is to prepare the voltage vector $\mathbf{u}(t)$ appearing in Eq. (163).

After preparing $\mathbf{u}(t)$, the time integral can be computed using circuitry 4620 that implements the thermal method in Figure 15. Namely, we consider the thermal method for RMS (root-mean-square) voltage measurement. In this method, a voltage source generates an unknown time-dependent voltage. This voltage source heats a resistor R_1. A digital device (e.g., a CPU) applies DC voltage to an equal resistance R_2 until both resistors reach the same temperature. The DC voltage is then equal to the RMS voltage of the voltage source. The temperature sensing can be carried out by two semiconductor diodes, which can be viewed as thermistors. The overall circuit diagram is shown in Fig. 19, and also includes an op amp. Applying this subroutine to the voltage source $u_i(t)$, which is the ith component of the voltage vector $\mathbf{u}(t)$, yields the subroutine

$$a_i = \sqrt{\int_0^1 (u_i(t))^2 dt}\,. \tag{163}$$

More circuitry 4630 applies the subroutine in Fig. 15 to the vector $\mathbf{a} = \{a_i\}$ and returns the norm

squared of the vector, giving an output voltage proportional to:

$$\sum_i a_i^2 = \sum_i \int_0^1 (u_i(t))^2 dt = \int_0^1 \|\mathbf{u}(t)\|_2^2 dt \tag{164}$$

An analog-to-digital converter 4640 converts this voltage to a digital value. The system can perform this same procedure for many different samples (i.e., many different k). A digital device 4650 averages the results over the different samples then returns the value for $\widehat{\lambda}_1$.

30 Training the Bayesian Neural Network

The Loss Evaluator discussed above can be used as a subroutine in the training process for a Bayesian neural network. This training process takes the form of a hybrid analog-digital process involving back-and-forth communication between the analog device (including the HLN, WD, PDN, and LE) and a digital processor. The analog device evaluates the loss function for a fixed value of the parameters, and then sends this loss function value to the digital processor. The digital processor then updates the parameters to new values, based on an optimization routine. This optimization routine could be chosen in a variety of ways, including gradient free methods and other standard methods. The new values of the parameters are fed back to the analog device, which then determines the new value of the loss function, and the process repeats. In this process, DACs and ADCs can be used to convert the parameter values to analog voltages and to convert the loss function values to digital signals, respectively.

An alternative approach is to use the analog device to compute gradients, and then feed these gradients to the digital process, which implements a gradient descent optimization routine. Computing gradients with the analog device could be done by adopting the adjoint sensitivity method. The adjoint sensitivity method is discussed in detail below. Moreover, this adjoint sensitivity method can be carried out on an analog device in the context of a neural ODE. Therefore, the electrical circuits presented below provide inspiration for how to perform the adjoint sensitivity method on our Bayesian Neural Network. Specifically, the methods below can be extended to the case of neural SDEs.

31 Byproduct technology: Analog Neural ODE

31.1 Overview

A subroutine of the thermodynamic AI system for Bayesian deep learning is an analog neural ODE. The analog neural ODE corresponds to/represents the Hidden Layer Network discussed in Section 25.3.

Neural networks represented by or that implement neural ODEs have a wide range of applications, both in supervised machine learning and in fitting time-series data. In this sense, our architecture for an analog neural ODE has applications beyond Bayesian deep learning and is relevant to deep learning in general as well as for fitting time-series data. Section 32 elaborates on the application to fitting time-series data.

Recall that an architecture for executing an analog neural ODE is shown in Figures 32, 33, 10, and 34. Figure 32 shows the unit cell. Figure 33 shows how the unit cells may be coupled. Figure 10 shows the connectivity structure. Figure 34 shows how to make an augmented version of the analog neural neural network for implementing an augmented neural ODE by expanding the space, i.e., increasing the number of unit cells.

The free parameters in the analog neural network for a neural ODE are voltages α, β, and $\mathbf{\Gamma}$, as illustrated in Figure 39. In general, these free parameters do not have to come from a weight diffuser (as

Fig. 30 shows), but instead can be supplied by a digital processor such as a CPU or FPGA. Alternatively, these parameters could be supplied by an analog device that evolves the weights over time but does not add stochastic noise, in which case the device acts as a weight evolver (instead of a weight diffuser).

31.2 The adjoint sensitivity method for computing gradients

The adjoint sensitivity method provides one way to compute the gradient of neural ODEs. Here, the gradient refers to the derivative of the loss function $\mathcal{L}$ with respect to the parameters θ, denoted $\frac{\partial \mathcal{L}}{\partial \theta}$.

This method involves:

- Running the neural network, which describes the evolution of the hidden layer values $h(t)$, backwards in time;
- Running a second differential equation backwards in time for the adjoint variable $a(t)$, where $a(t) = \frac{\partial \mathcal{L}}{\partial h}$; and
- Running a third differential equation backwards in time for the gradient $g(t) = \frac{\partial \mathcal{L}}{\partial \theta}$ with respect to the parameters θ.

The variables $h(t)$, $a(t)$, and $g(t)$ evolve together according to a system of coupled differential equations. This system has the form:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{h}\\ \mathbf{a}\\ \mathbf{g}\end{bmatrix} = \begin{bmatrix}\mathbf{f}(t,h,\theta)\\ -\mathbf{a}\frac{\partial f}{\partial h}\\ -\mathbf{a}\frac{\partial f}{\partial \theta}\end{bmatrix} \tag{165}$$

Here, the variable t runs backwards, say from $t = 1$ to $t = 0$. If we want a differential equation that runs forwards in time, then we can make the variable transformation $\tau = 1 - t$. Making this transformation gives:

$$\frac{d}{d\tau}\begin{bmatrix}\mathbf{h}\\ \mathbf{a}\\ \mathbf{g}\end{bmatrix} = \begin{bmatrix}-\mathbf{f}(1-\tau,h,\theta)\\ \mathbf{a}\frac{\partial f}{\partial h}\\ \mathbf{a}\frac{\partial f}{\partial \theta}\end{bmatrix} \tag{166}$$

where τ runs forwards from $\tau = 0$ to $\tau = 1$.

31.3 Computing gradients with analog approach

Consider an analog approach to the adjoint sensitivity method. The analog physical device can implement the system of differential equations in Eq. (166), with τ corresponding to the physical time over which the analog system evolves.

Achieving this goal involves introducing electrical circuits for the time evolution of $\mathbf{a}$ and $\mathbf{g}$. It also involves reversing the direction of the drift on $\mathbf{h}$, since a minus sign appears in the top line of Eq. (166), in contrast to Eq. (117). What follows covers these three subroutines:

1. Reversing the sign of the drift of $\mathbf{h}$;
2. Evolving $\mathbf{a}$ in time with analog circuitry; and
3. Computing $\mathbf{g}$ with analog circuitry.

31.4 Reversing the drift of h

Having the flexibility to apply a minus sign to the drift of $\mathbf{h}$ involves modifying the unit cell in the analog neural network. The unit cell in Fig. 32 does not allow for this possibility. This can be seen from the differential equation in Eq. (132), since, for example, the coefficient $\mathbf{J}$ can only have one particular sign.

Adding this additional flexibility (of applying a minus sign to the drift) can be done with a circuit such as the one in Fig. 47. Here, a voltage mixer is used to mix in a voltage source $k(t)$. If $k(t) = k_{\text{forward}}(t)$ for the neural network running in forwards time, then $k(t) = -k_{\text{forward}}(1-t)$ for the case of running the neural network in backwards time. Hence, this provides a means to flip the sign of the drift term in the analog neural ODE of the analog neural network.

31.5 The Adjoint Device

Next consider introducing circuits for the time evolution of $\mathbf{a}$. The circuit for evolving $\mathbf{a}$ is called the adjoint device.

Figure 48 shows a possible architecture for the adjoint device. Each unit cell is composed of a capacitor C_j^a and resistor R_j^a in series. In the adjoint device, there are N unit cells in total, the same number of unit cells as those in the analog device for executing the neural ODE.

In addition, the unit cells may be coupled by a resistive bridge with a resistance $R_{jj'}^a$, as shown in Figure 48. In principle, each unit cell can be connected to every other unit cell (full connectivity). It is also possible to restrict this connectivity such that the unit cells are connected to a lesser degree.

The voltages across the capacitors, denoted by the vector $\mathbf{v}^a$, represent the state variable. In other words, we encode the adjoint variable $\mathbf{a}$ in $\mathbf{v}^a$. Then, the state variable evolves over time according to the following differential equation:

$$\dot{\mathbf{v}}^a = -(\mathbf{C}^a)^{-1}\mathbf{J}^a\mathbf{v}^a = \mathbf{A}\mathbf{v}^a, \tag{167}$$

where

$$\dot{\mathbf{v}}^a \equiv (\frac{dv_1^a}{dt}, \frac{dv_2^a}{dt}, ..., \frac{dv_N^a}{dt}),\ \mathbf{C}^a \equiv \text{diag}(C_1^a, C_2^a, ..., C_N^a),$$
$$\mathbf{v}^a \equiv (v_1^a, v_2^a, ..., v_N^a),\ \mathbf{A} = -(\mathbf{C}^a)^{-1}\mathbf{J}^a.$$

In addition the matrix $\mathbf{J}^a$ has elements given by

$$\mathbf{J}_{jj'}^a = \begin{cases} \frac{1}{R_j^a} + \sum_k \frac{1}{R_{jk}^a} & \text{if } j = j' \\ -\frac{1}{R_{jj'}^a} & \text{if } j \neq j' \end{cases} \tag{168}$$

where we assume full connectivity between the unit cells (although this can be extended to partial connectivity).

Eq. (167) has the correct form provided that the $\mathbf{A}$ matrix can be related to the matrix $\frac{\partial f}{\partial h}$. This can be accomplished by encoding $\frac{\partial f}{\partial h}$ in the matrix $\mathbf{A}$.

One approach would be to compute $\frac{\partial f}{\partial h}$ digitally. Consider computing $\frac{\partial f}{\partial h}$ digitally at each time step (e.g., with automatic differentiation). A DAC can convert the digital signals associated with this matrix into analog voltages. The resulting analog voltages can be used to determine the various resistances R_j^a and $R_{jj'}^a$ of the resistors shown in Figure 48. These resistors can be physically implemented as voltage-controlled elements, such as transistors whose gate voltages controls the resistance. Therefore, we can take the analog voltages associated with the matrix elements of $\frac{\partial f}{\partial h}$ and apply them as gate voltages to the transistors that determine the resistances R_j^a and $R_{jj'}^a$. This encodes the matrix $\frac{\partial f}{\partial h}$ into the matrix $\mathbf{A}$.

A different approach is to compute $\frac{\partial f}{\partial h}$ in an analog fashion with an analytical model for f. For example, we may know that the expression is given by $f = \mathbf{C}^{-1}[\mathbf{J}_s\alpha - \mathbf{J}\mathbf{v} - \mathbf{I}_{\text{NL}}]$, which appears in Eq. (122). We can then analytically differentiate this expression with respect to $\mathbf{h}$ (or in this case $\mathbf{v}$,

since $\mathbf{v}$ is a surrogate variable for $\mathbf{h}$). Hence, suppose we have an analytical formula for $\frac{\partial f}{\partial h} = g(h)$, which is some function $q(h)$ of h. An analog circuit can implement $q(h)$ (e.g., if $q(h)$ is an affine function then one can implement matrix-vector multiplication with an analog circuit in a straightforward manner using a layer of resistors). This involves reading off the vector h in real time during the evolution of the neural network and passing it through the circuit that implements $q(h)$. The output of this function can be fed to the adjoint device with appropriate gate voltages applied to the transistors used to represent the resistors R_j^a and $R_{jj'}^a$.

31.6 Circuits for computing g

To compute the gradient $\mathbf{g}$, we first obtain the matrix elements of $\frac{\partial f}{\partial \theta}$. Once again, obtaining this matrix can be done with digital numerical methods (like automatic differentiation) followed by digital-to-analog conversion. Alternatively, it can be done by having an analytical expression for f, then analytically computing $\frac{\partial f}{\partial \theta} = \tilde{q}(h)$ and expressing it as some function $\tilde{q}(h)$, and then passing the analog vector $\mathbf{h}$ (namely its surrogate $\mathbf{v}$) through a circuit that implements $\tilde{q}(h)$ (e.g., a circuit that does matrix-vector multiplication if $\tilde{q}(h)$ is affine).

Next, during the time evolution of the adjoint device, the vector $\mathbf{a}$ can be read by picking off the voltages across the capacitors in the adjoint device. This vector can be fed into an analog circuit that performs matrix-vector multiplication (e.g., using a layer of resistors). This matrix-vector multiplication can involve multiplying $\mathbf{a}$ by an analog version of the matrix $\frac{\partial f}{\partial \theta}$, which was obtained in the previous step.

Now that the matrix-vector product $\mathbf{a}\frac{\partial f}{\partial \theta}$ has been obtained as an analog voltage, it can be integrated over time. This can involve a set of integrator circuit. Figure 49 shows a circuit diagram for an integrator, i.e., a circuit that computes the time integral of a voltage signal. There can be an integrator circuit for each component of the gradient g. The output of this set of integrator circuit is be the gradient g.

While the above approach is analog, some operations can be performed digitally. For example, the matrix-vector multiplication $\mathbf{a}\frac{\partial f}{\partial \theta}$ can be performed digitally by first passing $\mathbf{a}$ through an ADC, then digitally integrating this matrix-vector product.

31.7 Training the analog neural network

The method above can be used for computing gradients with respect to the parameters θ of the analog neural ODE that represents the analog neural network. In practice, these parameters values can be stored digitally in memory of or coupled to a CPU or FPGA, and then supplied to the analog circuit as needed.

The training process can therefore correspond to an optimization routine that involves a hybrid analog-digital feedback loop. Here the digital device supplies the values of the parameters θ, then the analog device computes the gradient, then the digital device provides new values for the parameters after doing a gradient descent step, and the process repeats.

DACs and ADCs can be used to convert the parameter values to analog voltages and to convert the gradient values to digital signals, respectively. Hence, this corresponds to using a gradient-descent approach to training the analog neural network.

Adjoint sensitivity methods are not needed in a gradient-free approach to training the analog neural network. Instead, an analog neural network can be trained using a loss function based on the outputs of the analog neural network. Then, this loss function value can be used in the context of a gradient-free optimization routine that is facilitated by a digital device.

32 Byproduct application: fitting time-series data

32.1 Overview

Time-series data provide an important application relevant to financial analysis, market prediction, epidemiology, and medical data analysis. In many cases, data at particular time points may be at irregular time intervals. In these cases, it can be helpful to have a model that makes predictions at all times and hence interpolates between the datapoints and extrapolates beyond the data, e.g., to make predictions about the future where no data is available.

Discrete neural networks, such as recurrent neural networks, have been used in the past for interpolating and extrapolating time-series data. However, neural networks that implement or obey latent ODEs have been shown to outperform recurrent neural networks at this task.

Latent ODEs are continuous time models that are essentially the same as neural ODEs, although the different names are used to distinguish the application. Namely, a neural ODE (latent ODE) represents a neural network used for supervised machine learning (fitting time-series data). A latent ODE can be viewed as a parameterized ODE, where the corresponding neural network has parameters trained to fit the time-series data (according to some loss function).

32.2 Analog Latent ODEs

Section 31 presents an analog architecture for implementing neural ODEs. We can employ this analog architecture to carry out a subroutine of a latent ODE model for fitting time-series data.

Figure 50 provides a schematic diagram for our analog implementation of a latent ODE. The analog latent ODE implementation in Fig. 50 has three components or subsystems:

1. an encoder;
2. an analog neural ODE processor; and
3. a decoder.

The training data are provided as observations from some time series. These time-series observations are fed into an encoder. The encoder has free parameters that can be trained. For example, the encoder could be a recurrent neural network. The output of the encoder can be the initial vector $\mathbf{h}(0)$ of the hidden layer values, or the output could be a probability distribution from which $\mathbf{h}(0)$ is sampled. If the encoder is stored on a digital device, its output can pass through a DAC, after which it can be fed as an analog signal to the analog neural ODE.

The analog neural ODE processor, which is presented in Sec. 31, provides the latent space for the latent ODE. This latent space is initialized to $\mathbf{h}(0)$ by the encoder. Then then hidden layer values evolve over time according to the differential equation that describes the analog neural ODE. Recall that Figures 32, 33, 10, and 34 provide details for how the analog neural ODE processor can be constructed.

The hidden layer values $h(t_k)$ can be read off at a set $\{t_k\}$ of various times, by measuring the voltages on the capacitors in the analog neural ODE processor. This set $\{h(t_k)\}$ of values can be fed to a decoder. The decoder could be a neural network that is stored on a digital device. In this case, ADCs can digitize the analog signals associated with $\{h(t_k)\}$ and feed the resulting digital signals to the decoder. The decoder can have free parameters that to be trained. The decoder can be probabilistic, such that the outputs predicted by the decoder are a probabilistic function of the hidden layer values $\{h(t_k)\}$. A benefit of making the decoder probabilistic is that it can be used to simulate noisy processes or stochastic processes.

The outputs of the decoder correspond to predictions that the latent ODE model makes for the true time series. These predictions can go beyond the time interval associated with the observations, in which case the predictions correspond to extrapolated values.

A training process occurs where the parameters of the encoder, the decoder, and the analog neural ODE processor are optimized in order to minimize or maximize a loss function. This essentially corresponds to fitting the time-series data. For example, a loss function based on the evidence-based lower bound (ELBO) can be employed in this training process, or other loss functions are also possible. A gradient based approach can be taken, and the adjoint sensitivity method discussed in Sec. 31 can be employed for computing gradients.

32.3 Extension to analog latent SDEs

The framework and architecture presented in Figure 30 can be extended as follows. An analog neural SDE processor can replace the analog neural ODE processor for the latent space used in Figure 30. Replacing the analog neural ODE with an analog neural SDE results in an analog latent SDE. The analog latent SDE can be used to model time-series data that are generated by stochastic processes. This has useful applications in financial and market analysis.

Fig. 31 illustrates an analog neural SDE processor with a stochastic noise source in each unit cell of the architecture for the analog neural ODE processor. The voltage source B_j denotes a stochastic noise source. Aside from this additional voltage source, the rest of the architecture of the analog neural SDE processor could be the same as that of the analog neural ODE processor.

An analog neural SDE processor inserted into the latent space shown in Fig. 30 in place of the analog neural ODE processor could be employed to fit and extrapolate time-series data from sources that are stochastic in nature.

33 General Framework

Here we provide a general framework that encompasses multiple applications. This includes the applications discussed previously (diffusion models and Bayesian deep learning), as well as other applications.

33.1 Unification of Thermodynamic AI algorithms

Our aim here is to present a thermodynamic hardware paradigm that is relevant to multiple AI applications. A stepping stone towards this goal is to mathematically unify different AI applications under the same framework. The applications we consider are illustrated in Figure 32 and also enumerated below:

1. Generative diffusion models
2. Bayesian neural networks
3. Monte Carlo inference
4. Annealing
5. Time series forecasting

Through careful consideration, we manage to formulate a mathematical framework that encompasses all of the aforementioned algorithms as special cases. We say that these algorithms belong to a class called *Thermodynamic AI algorithms.*

At a conceptual level, we can define Thermodynamic AI algorithms as algorithms consisting of at least two subroutines:

1. A subroutine in which a stochastic differential equation (SDE) is evolved over time.

2. A subroutine in which a Maxwell's demon observes the state variable in the SDE and applies a drift term in response.

At the mathematical level, we propose that Thermodynamic AI algorithms are ones that simulate or implement the following set of equations (or some subset of them):

$$d\mathbf{p} = [\mathbf{f} - BM^{-1}\mathbf{p}]dt + Dd\mathbf{w} \tag{169}$$

$$d\mathbf{x} = M^{-1}\mathbf{p}\,dt \tag{170}$$

$$\mathbf{f} = -\nabla_{\mathbf{x}} U_\theta \tag{171}$$

One can see that these correspond to Newton's laws of motion, with the addition of diffusion and friction. In these equations, $\mathbf{p}$, $\mathbf{x}$, and $\mathbf{f}$ respectively are the momentum, position, and force. The matrices M, D, and B are hyperparameters, with M being the mass matrix and D being the diffusion matrix. The $d\mathbf{w}$ term is a Wiener process. Finally, U_θ is a (trainable) potential energy function. Typically, much of the application-specific information, regarding the task to be solved, is encoded in the potential energy function U_θ. Note that for readability we omit the dependencies of the variables on time t and space $\mathbf{x}$ in the above equations.

This unification is crucial to developing a hardware paradigm that is broadly applicable to many AI algorithms. In what follows, we will describe the fundamental building blocks for this hardware paradigm.

33.2 Fundamental building blocks

	classical	quantum	probabilistic	thermodynamic
discrete	bit	qubit	p-bit	s-bit
continuous	mode	qumode	p-mode	s-mode

Let us now discuss the fundamental building blocks of thermodynamic AI hardware. As the name "thermodynamic" suggests, a thermodynamic system is inherently dynamic in nature. Therefore, the fundamental building blocks should also be dynamic. This is contrast to classical bits or qubits, where the state of the system ideally remains fixed unless it is actively changed by gates. The thermodynamic building block should passively and naturally evolve over time, even without the application of gates.

But what dynamical process should it follow? A reasonable proposal is a stochastic Markov process. Naturally this should be continuous in time, since no time point is more special than any other time point. Hence, the discrete building block, which we call an s-bit, would follow a continuous-time Markov chain (CTMC). Here the "s" in s-bit stands for stochastic.

For the continuous building block, which we call an s-mode, the natural analog would be a Brownian motion (also known as a Wiener process). One can impose that this process is a Martingale (which is typically assumed for Brownian motion), which means that it has no bias. We use s-unit as a generic term to encompass both s-bits and s-modes.

For comparison, one can consider the fundamental building blocks of a probabilistic system, which are p-bits (p-modes) for the discrete case (continuous case). The p-bit can be thought of as a random number generator, which either generates 0 or 1 at random. The analog of this in the continuous case, which we call a p-mode, could be a random number generator that generates a real number according to a Gaussian distribution with zero mean and some variance.

Although p-bits (p-modes) are clearly different than s-bits (s-modes), there is some connection between them. Namely, one could think of the latter as the time integral of the former. For example, one

could view an s-mode as the Ito integral of a p-mode. In this sense, one could experimentally construct s-bits (s-modes) if one has access to both p-bits (p-modes) as well as a time-integrator apparatus.

33.3 Physical realizations of s-modes

33.3.1 Individual s-modes

An s-mode represents a continuous stochastic variable whose dynamics are governed by drift, diffusion, or other physical processes (akin to a Brownian particle). At the heart of any physical implementation of such a variable will be a source of stochasticity. A natural starting point for implementing thermodynamic AI hardware is analog electrical circuits, as these circuits have inherent fluctuations that could be harnessed for computation.

The most ubiquitous source of noise in electrical circuits is thermal noise. Thermal noise, also called Johnson-Nyquist noise, comes from the random thermal agitation of the charge carriers in a conductor, resulting in fluctuations in voltage or current inside the conductor. Thermal noise is Gaussian and has a flat frequency spectrum (white noise) with fluctuations in the voltage of standard deviation

$$v_{\mathrm{tn}} = \sqrt{4k_B T R \Delta f}, \tag{172}$$

where R is the resistance of the conductor, k_B is the Boltzmann constant, T is the absolute temperature and Δf is the frequency bandwidth. The amplitude of the voltage fluctuations can be controlled by changing the temperature or the resistance. In practice, a thermal noise source can be implemented using a large resistor in series with a voltage amplifier.

Another type of electrical noise is shot noise. Shot noise arises from the discrete nature of charge carriers and from the fact that the probability of a charge carrier crossing a point in a conductor at any time is random. This effect is particularly important in semiconductor junctions where the charge carriers should overcome a potential barrier to conduct a current. The probability of a charge carrier passing over the potential barrier is an independent random event. This induces fluctuations in the current through the junction. Shot noise is Gaussian and has a flat frequency spectrum (white noise) with fluctuations in the current of standard deviation

$$I_{\mathrm{tn}} = \sqrt{2q|I|\Delta f}, \tag{173}$$

where I is the current through the junction, q is the electron charge and Δf is the frequency bandwidth. The amplitude of the current fluctuations can be controlled by changing the magnitude of the DC current passing through the junction. In practice, a source of shot noise would be implemented using a pn diode (for example, a Zener diode in reverse bias configuration) in series with a controllable current source.

In general, any physical implementation of s-modes should have the amplitude of its stochasticity be independently controllable with respect to the other system parameters. We have seen that electrical thermal and shot noise, for example, both have tuning knobs to control the amplitude of the noise to some extent.

In addition, one must ensure that the amplitude of the fluctuations be compatible, i.e. measurable, with the rest of the system. Thermal and shot noise sources typically have voltage fluctuations of the order of a few μV or less. For these fluctuations to be measurable by on-chip analog-to-digital converters measuring voltages on the order of hundreds of mV, amplification will be necessary. This amplification can be done using single- or multi-stage voltage amplifiers. Variable-gain amplifiers can also let one independently control the amplitude of the fluctuations.

The s-mode can be represented through the dynamics of any degree of freedom of an electrical circuit.

If we chose the voltage on a particular node in the circuit as our degree of freedom of choice, a simple stochastic voltage noise source plays the role of the s-mode. This can be realized by using a noisy resistor at non-zero temperature. The circuit schematic in Fig. 53 shows the typical equivalent noise model for a noisy resistor composed of a stochastic voltage noise source, $\delta v(t)$, in series with an ideal (non-noisy) resistor of resistance R. The inherent terminal capacitance, C, of the resistor is also added to the equivalent resistor model. We chose to use the voltage on node 1 (labeled simply as $v(t)$ here) as our s-mode. The dynamics of the s-mode in this case, obeys the following SDE model:

$$-\frac{\mathrm{d}v(t)}{\mathrm{d}t} = \frac{v(t) + \delta v(t)}{RC}. \tag{174}$$

The voltage fluctuations in Eq. (174) have the usual Gaussian white noise properties $\langle \delta v(t) \rangle = 0$ and $\langle \delta v(t) \delta v(t') \rangle = 2k_B T R \delta(t - t')$ where, $\langle\ \rangle$ represents statistical averaging and where $\delta(t - t')$ represents the Dirac delta function. We notice the form of the SDE comprises a drift term proportional to $v(t)$ and a diffusion or stochastic term proportional to $\delta v(t)$.

In order for the s-modes to be more useful for computation, their inherent stochastic dynamics must be constrained to the properties of an algorithm. One can, for example add a drift term to the dynamics of the s-mode by adding a capacitor in series with the noisy resistor. In general, one can add other electrical components, such as inductors or non-linear elements, to further constrain the evolution of the s-mode.

33.3.2 Coupling s-modes

When building systems of many s-modes, one will most likely wish to introduce some form of coupling between them to express correlations and geometric constraints. Again, the medium of analog electrical circuits presents a natural option for the coupling of s-modes.

As a first example, two circuits of the type described in section 33.3.1 could be coupled through a resistor, as pictured in the upper panel of Fig. 54. The coupled s-modes, represented by the voltage on nodes 1 and 2, are then coupled through their drift terms as (we omit the time dependencies for readability)

$$-\frac{v_1}{t} = \frac{v_1 + \delta v_1}{R_1 C_1} - \frac{v_2 - v_1}{R_{12} C_1}, \tag{175}$$

$$-\frac{v_2}{t} = \frac{v_2 + \delta v_2}{R_2 C_2} - \frac{v_1 - v_2}{R_{12} C_2}. \tag{176}$$

To simplify the notation, this system of coupled equations can be written in matrix form as follows

$$-\dot{\mathbf{v}} = \mathbf{C}^{-1} \left(\mathbf{J}\mathbf{v} + \mathbf{R}^{-1} \delta \mathbf{v} \right), \tag{177}$$

where $\dot{\mathbf{v}} \equiv \frac{}{t}\mathbf{v}$ and

$$\mathbf{v} \equiv \begin{bmatrix} v_1 \\ v_2 \end{bmatrix},\ \mathbf{C} \equiv \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix},\ \mathbf{J} \equiv \begin{bmatrix} \frac{1}{R_1} + \frac{1}{R_{12}} & -\frac{1}{R_{12}} \\ -\frac{1}{R_{12}} & \frac{1}{R_2} + \frac{1}{R_{12}} \end{bmatrix},\ \mathbf{R} \equiv \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix},\ \delta \mathbf{v} \equiv \begin{bmatrix} \delta v_1 \\ \delta v_2 \end{bmatrix}.$$

Where we have introduced the self-resistance matrix $\mathbf{R}$, the capacitance matrix $\mathbf{C}$, and the conductance matrix $\mathbf{J}$.

A second method of coupling two s-modes together is by using a capacitor as a the coupling element, as pictured in the lower panel of Fig. 54. In this configuration, the two coupled s-modes, represented by the voltage on nodes 1 and 2, have drift and diffusion coupling as (we omit the time dependencies for

readability)

$$-\frac{v_1}{t} = \frac{v_1 + \delta v_1}{R_1(C_1 + C_{12})} + \frac{C_{12}(v_2 + \delta v_2)}{R_2(C_1 + C_{12})(C_2 + C_{12})} \tag{178}$$

$$-\frac{v_2}{t} = \frac{v_2 + \delta v_2}{R_2(C_2 + C_{12})} + \frac{C_{12}(v_1 + \delta v_1)}{R_1(C_1 + C_{12})(C_2 + C_{12})}. \tag{179}$$

To simplify the notation, using the same notation as in Eq. (177), this system of coupled equations can be written in matrix form as follows

$$-\dot{\mathbf{v}} = \mathbf{C}^{-1}\mathbf{R}^{-1}\left(\mathbf{v} + \delta\mathbf{v}\right), \tag{180}$$

where the capacitance matrix is

$$\mathbf{C} \equiv \begin{bmatrix} C_1 + C_{12} & -C_{12} \\ -C_{12} & C_2 + C_{12} \end{bmatrix}.$$

33.4 Maxwell's Demon

Maxwell's Demon is a thermodynamic concept that is similar to refrigeration, as it allows a system's entropy to be reduced over time by interacting with an external system. Here, the demon acts as an intelligent observer who regularly gathers data from (i.e., measures) the system, and based on the gathered information, the demon performs some action on the system. The classic example involves a gaseous mixture and a physical barrier as illustrated in Figure 11, although it can be implemented by various physical means including with electrical ciruits.

We argue that a Maxwell Demon is both: (1) A key component of Thermodynamic AI systems due to the complex entropy dynamics required for AI applications, and (2) Straightforward to implement in practice for several different hardware architectures.

Regarding the first point, AI applications like Bayesian inference aim to approximate a posterior distribution, and it is known that such posteriors can be extremely complicated and multi-modal. Similarly, generative modeling is intended to handle arbitrary data distributions. Hence, producing only Gaussian distributions, as an isolated s-mode system would do, will not suffice for these applications. Regarding the second point, we discuss below how one can implement a Maxwell's Demon in hardware.

33.4.1 Digital Maxwell's Demon

We regard the Maxwell's Demon (MD) as a hardware component of Thermodynamic AI systems. One can construct the MD device in a variety of ways, with digital or analog approaches. A digital MD system is quite simple to describe. Namely, it can correspond to a neural network that is stored on a digital processor, such as a CPU or FPGA. In this case, one would need to communicate the state vector $\mathbf{v}$ to the digital processor (to be the input to the neural network), and then communicate the proposed action of the Maxwell's Demon (i.e., the output of the neural network) back to the thermodynamic hardware. Hence, one simply needs a means to interconvert signals between the thermodynamic hardware and the digital processor. This is illustrated in diagrams such as Fig. 12 or Fig. 41, with the interconversion shown as analog-to-digital converters (ADCs) and digital-to-analog converters (DACs).

33.4.2 Analog Maxwell's Demon

An analog MD device could allow one to integrate it more closely to the rest of the thermodynamic hardware. Moreover, this could allow one to avoid interconverting signals. Here we describe a force-

based approach for the MD device, with alternative approaches being possible (such as the total derivative approach discussed in previous sections).

Recall from Newtonian mechanics that the force $\mathbf{f}(\mathbf{x})$ on a system at position $\mathbf{x}$ is given by the gradient of the potential energy function, $U(\mathbf{x})$, and the momentum $\mathbf{p}$ is proportional to the time derivative of the position:

$$\mathbf{f}(\mathbf{x}) = -\nabla_{\mathbf{x}} U(\mathbf{x}), \qquad \frac{d\mathbf{x}}{dt} = -M^{-1}\mathbf{p}\,. \tag{181}$$

Here M is the mass matrix. Equation (181) provides a recipe for how to construct an MD device. The idea would be to view the momentum $\mathbf{p}$ as the state vector associated with the s-mode system. Hence, the momentum will evolve over time according to the SDE equation associated with the s-mode system, such as Eq. (177). At a given time t, the MD system will take the momentum vector in as an input. Then the MD system will output a force that is a function of both the time t and the momentum $\mathbf{p}(t)$,

$$\text{Input from s-modes: } \mathbf{p}(t), \quad \text{Output to s-modes: } \mathbf{f}(t, \mathbf{p}(t)) \tag{182}$$

Let us discuss how the MD device performs the mapping from input to output in Eq. (182). Suppose the MD device has a latent variable or hidden variable, which corresponds to the position vector $\mathbf{x}(t)$. The latent variable $\mathbf{x}(t)$ is stored inside the MD device's memory, and it evolves over time. Specifically, it evolves over time according to the differential equation in Eq. (181). Hence, for the latent variable, we have:

$$\text{Latent variable:} \quad \mathbf{x}(t) = \mathbf{x}(0) - \int_{t'=0}^{t'=t} M^{-1}\mathbf{p}(t')dt'\,, \tag{183}$$

where $\mathbf{x}(0)$ is an initial starting point for the latent variable.

The MD device also stores a potential energy function $U_\theta(t, \mathbf{x}(t))$. For generality, we allow this potential energy function to be time-dependent. This time dependence is important for certain applications such as annealing, where one wishes to vary the potential energy function over time. In addition, we also allow the potential energy function to depend on a set of parameters θ. These parameters are trainable and will be trained during the training process, such as in diffusion models. Hence, the potential energy function represents the trainable portion of the MD device,

$$\text{Trainable Potential Energy Function:} \quad U_\theta(t, \mathbf{x}(t))\,. \tag{184}$$

Internally, the MD device combines (183) and (184) to produce a force. Specifically, the MD device employs Eq. (181) and computes the gradient of the potential energy function at time t and location $\mathbf{x}(t)$, as illustrated in Figure 55. The result of this computation is the output of the MD device:

$$\text{Output:} \quad \mathbf{d}_\theta(t, \mathbf{v}(t)) = \mathbf{d}_\theta(t, \mathbf{p}(t)) = -\nabla_{\mathbf{x}} U_\theta(t, \mathbf{x}(t))\,. \tag{185}$$

33.5 Fitting diffusion models into the general framework

Fig. 53 shows how diffusion models and other applications fall under our general framework.

For diffusion models, changing variables $\tau = T - t$ in the reverse process makes it possible to rewrite the SDE equations as:

$$\mathrm{d}\mathbf{x} = f(t)\mathbf{x}\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}_t \quad \text{(Forward)} \tag{186}$$

$$\mathrm{d}\mathbf{x} = -f(T-\tau)\mathbf{x}\mathrm{d}\tau + g(T-\tau)^2\mathbf{s}_\theta(\mathbf{x}, T-\tau)\mathrm{d}\tau + g(T-\tau)\mathrm{d}\mathbf{w}_\tau \quad \text{(Reverse)} \tag{187}$$

In this case, both t and τ run forward in time, i.e., dt and $d\tau$ are positive increments in these equations. Note that $\mathrm{d}\mathbf{w}_\tau$ appears in this equation since $\mathrm{d}\mathbf{w}_\tau = \mathrm{d}\overline{\mathbf{w}}_t$. Since the time variables in these equations run forward in time, then they can be directly implemented in physical analog hardware, since time runs forward in the physical world.

The diffusion models can fit into our framework using the following mapping:

$$\text{(diffusion process)} \leftrightarrow \text{(s-mode device)} \tag{188}$$

$$\text{(score network)} \leftrightarrow \text{(Maxwell's demon device)} \tag{189}$$

The mathematical diffusion process in diffusion models can be mapped to the physical diffusion process in the s-mode device. Similarly, the score vector outputted by the score network corresponds to the vector $\mathbf{d}(t, \mathbf{v}(t))$ outputted by the MD device. Hence, diffusion models fit in our general framework for Thermodynamic AI systems.

33.6 Fitting Bayesian Deep Learning into the general framework

Bayesian deep learning, discussed above in Section 21), fits into our general framework as follows. We make the following mapping in order to fit this into our framework:

$$\text{(weight diffuser)} \leftrightarrow \text{(s-mode device)} \tag{190}$$

$$\text{(posterior drift network)} \leftrightarrow \text{(Maxwell's demon device)} \tag{191}$$

The weight diffuser corresponds to the s-mode device, and the posterior drift network corresponds to the Maxwell's demon device. The weight diffuser uses s-mode dynamics to sample weight trajectories w_t, which are then imported into the HLN. The Maxwell's demon is used to produce complex s-mode dynamics that produce weight trajectories according to a posterior distribution.

33.7 Fitting annealing into the general framework

Annealing processes, such as simulating annealing, fit into our general framework as follows. Suppose that we have an optimization problem, where the loss function $\mathcal{L}(\mathbf{x})$ of interest is a continuous function of an N-dimensional state variable $\mathbf{x}$. Here, optimizing over $\mathbf{x}$ solves the optimization problem in the context of simulated annealing. In this setting, consider proposing a coupled system of equations for the state variable $\mathbf{x}(t)$ and an auxiliary variable $\mathbf{p}(t)$:

$$d\mathbf{x}(t) = \mathbf{p}(t)dt \tag{192}$$

$$d\mathbf{p}(t) = -\nabla\mathcal{L}(\mathbf{x})dt - \frac{1}{2}D\,\mathbf{p}(t)dt + S\,d\mathbf{W} \tag{193}$$

Here, $\mathbf{W}$ is an N-dimensional Brownian motion, S is an $N \times N$ dimensional lower-triangular matrix, and $D = SS^\intercal$. The first differential equation is called the optimization ODE, and Eq. (193) is called the auxiliary SDE. The dynamics on the state variable $\mathbf{x}(t)$ are effectively stochastic, as it is coupled to the auxiliary variable evolving via the auxiliary SDE. The equations in (192) and (193) describe a simulated annealing process.

Equations (192) and (193) are special cases of our general framework, given in Eqs. (169) and (170),

for Thermodynamic AI hardware. Specifically, we have the following mapping to our hardware:

$$\text{(auxiliary SDE)} \leftrightarrow \text{(s-mode device)} \tag{194}$$

$$\text{(optimization ODE)} \leftrightarrow \text{(latent variable evolution in Maxwell's demon device)} \tag{195}$$

The idea is that the auxiliary SDE describing the evolution of $\mathbf{p}$ can be performed on the s-mode device. In addition, $-\nabla\mathcal{L}(\mathbf{x})$ corresponds to the vector $\mathbf{d}$ output by the Maxwell's demon in our hardware. The optimization ODE then maps onto the evolution of the latent variable in the Maxwell's demon device. This employs a forced-based Maxwell's demon, as discussed above and shown Figure 55. Also, note that the mass matrix that appears in our framework is set to be the identity for this application: $M = I$.

34 Conclusion

While various inventive embodiments have been described and illustrated herein, those of ordinary skill in the art will readily envision a variety of other means and/or structures for performing the function and/or obtaining the results and/or one or more of the advantages described herein, and each of such variations and/or modifications is deemed to be within the scope of the inventive embodiments described herein. More generally, those skilled in the art will readily appreciate that all parameters, dimensions, materials, and configurations described herein are meant to be exemplary and that the actual parameters, dimensions, materials, and/or configurations will depend upon the specific application or applications for which the inventive teachings is/are used. Those skilled in the art will recognize or be able to ascertain, using no more than routine experimentation, many equivalents to the specific inventive embodiments described herein. It is, therefore, to be understood that the foregoing embodiments are presented by way of example only and that, within the scope of the appended claims and equivalents thereto, inventive embodiments may be practiced otherwise than as specifically described and claimed. Inventive embodiments of the present disclosure are directed to each individual feature, system, article, material, kit, and/or method described herein. In addition, any combination of two or more such features, systems, articles, materials, kits, and/or methods, if such features, systems, articles, materials, kits, and/or methods are not mutually inconsistent, is included within the inventive scope of the present disclosure.

Also, various inventive concepts may be embodied as one or more methods, of which an example has been provided. The acts performed as part of the method may be ordered in any suitable way. Accordingly, embodiments may be constructed in which acts are performed in an order different than illustrated, which may include performing some acts simultaneously, even though shown as sequential acts in illustrative embodiments.

All definitions, as defined and used herein, should be understood to control over dictionary definitions, definitions in documents incorporated by reference, and/or ordinary meanings of the defined terms.

The indefinite articles "a" and "an," as used herein in the specification and in the claims, unless clearly indicated to the contrary, should be understood to mean "at least one."

The phrase "and/or," as used herein in the specification and in the claims, should be understood to mean "either or both" of the components so conjoined, i.e., components that are conjunctively present in some cases and disjunctively present in other cases. Multiple components listed with "and/or" should be construed in the same fashion, i.e., "one or more" of the components so conjoined. Other components may optionally be present other than the components specifically identified by the "and/or" clause, whether related or unrelated to those components specifically identified. Thus, as a non-limiting example, a reference to "A and/or B", when used in conjunction with open-ended language such as "comprising" can refer, in one embodiment, to A only (optionally including components other than B); in another embodiment, to B only (optionally including components other than A); in yet another embodiment, to

both A and B (optionally including other components); etc.

As used herein in the specification and in the claims, "or" should be understood to have the same meaning as "and/or" as defined above. For example, when separating items in a list, "or" or "and/or" shall be interpreted as being inclusive, i.e., the inclusion of at least one, but also including more than one, of a number or list of components, and, optionally, additional unlisted items. Only terms clearly indicated to the contrary, such as "only one of" or "exactly one of," or, when used in the claims, "consisting of," will refer to the inclusion of exactly one component of a number or list of components. In general, the term "or" as used herein shall only be interpreted as indicating exclusive alternatives (i.e., "one or the other but not both") when preceded by terms of exclusivity, such as "either," "one of," "only one of," or "exactly one of." "Consisting essentially of," when used in the claims, shall have its ordinary meaning as used in the field of patent law.

As used herein in the specification and in the claims, the phrase "at least one," in reference to a list of one or more components, should be understood to mean at least one component selected from any one or more of the components in the list of components, but not necessarily including at least one of each and every component specifically listed within the list of components and not excluding any combinations of components in the list of components. This definition also allows that components may optionally be present other than the components specifically identified within the list of components to which the phrase "at least one" refers, whether related or unrelated to those components specifically identified. Thus, as a non-limiting example, "at least one of A and B" (or, equivalently, "at least one of A or B," or, equivalently "at least one of A and/or B") can refer, in one embodiment, to at least one, optionally including more than one, A, with no B present (and optionally including components other than B); in another embodiment, to at least one, optionally including more than one, B, with no A present (and optionally including components other than A); in yet another embodiment, to at least one, optionally including more than one, A, and at least one, optionally including more than one, B (and optionally including other components); etc.

In the claims, as well as in the specification above, all transitional phrases such as "comprising," "including," "carrying," "having," "containing," "involving," "holding," "composed of," and the like are to be understood to be open-ended, i.e., to mean including but not limited to. Only the transitional phrases "consisting of" and "consisting essentially of" shall be closed or semi-closed transitional phrases, respectively, as set forth in the United States Patent Office Manual of Patent Examining Procedures, Section 2111.03.

Claims

1. A system for executing a generative diffusion model on a dataset, the system comprising:

a physical system having a plurality of degrees of freedom, each degree of freedom in the plurality of degrees of freedom associated with a corresponding feature of the dataset and having a continuous state variable that evolves according to a corresponding differential equation having a tunable diffusion term and a tunable drift term.

2. The system of claim 1, further comprising:

a processor, operably coupled to the physical system, to reduce entropy of the continuous state variables so as to produce an output of the generative diffusion model.

3. The system of claim 2, wherein the processor is configured to reduce the entropy of the continuous state variables by reading off the continuous state variables and altering at least one of the tunable drift terms.

4. The system of claim 2, wherein the processor is configured to reduce the entropy of the continuous state variables by executing a score network.

5. The system of claim 2, wherein the processor is configured to be trained by an optimization routine that reduces a loss function.

6. The system of claim 5, wherein the physical system is configured to assist with estimating the loss function.

7. The system of claim 6, wherein the processor comprises analog circuitry configured to assist with estimating the loss function.

8. The system of claim 7, wherein estimating the loss function involves simultaneous time evolution of the physical system and of the analog circuitry in the processor.

9. The system of claim 8, wherein the loss function quantifies a degree of score matching.

10. The system of claim 1, wherein the degrees of freedom are physically coupled to each other according to a problem geometry associated with the dataset.

11. The system of claim 1, further comprising:

a function generator, operably coupled to the physical system, to multiply the tunable drift terms and tunable diffusion terms by arbitrary time-dependent functions so as to modify the differential equations governing evolution of the continuous state variables.

12. The system of claim 1, further comprising:

a digital device, operably coupled to the physical system, to upload the dataset to the degrees of freedom and to download new data corresponding to measurements of values of the continuous state variables after evolution of the continuous state variables.

13. The system of claim 1, wherein the physical system comprises a network of electrical circuits,

each electrical circuit in the network of electrical circuits providing a corresponding degree of freedom in the plurality of degrees of freedom.

14. The system of claim 13, wherein each electrical circuit in the network of electrical circuits comprises:

a capacitor having a charge or voltage encoding the continuous state variable;

a stochastic noise source, in series with the capacitor, to generate the tunable diffusion term; and

a resistor, in series with the capacitor, to generate the tunable drift term.

15. The system of claim 14, wherein the stochastic noise source is a thermal noise source and/or a shot noise source.

16. The system of claim 14, wherein each electrical circuit in the network of electrical circuits further comprises:

a first tunable voltage source, operably coupled to the stochastic noise source, to tune the tunable diffusion term of the corresponding differential equation; and

a second tunable voltage source, operably coupled to the resistor, to tune the tunable drift term of the corresponding differential equation.

17. The system of claim 14, wherein each electrical circuit in the network of electrical circuits further comprises:

a variable resistor, in parallel with the capacitor and having a variable resistance controlled by a tunable voltage source, to tune the tunable drift term of the differential equation; and

an amplifier, operably coupled to the stochastic noise source, to amplify an output of the stochastic noise source so as to tune the tunable diffusion term of the differential equation.

18. The system of claim 17, wherein the amplifier has a variable gain determined by an additional variable resistor whose resistance is controlled by an additional tunable voltage source.

19. The system of claim 13, further comprising:

switches coupling the electrical circuits in the network of electrical circuits according to a problem geometry associated with the dataset.

20. The system of claim 13, further comprising:

a processor, operably coupled to the network of electrical circuits, to reduce entropy of the continuous state variables so as to produce an output of the generative diffusion model.

21. The system of claim 20, wherein the processor is configured to reduce the entropy of the continuous state variables by reading off the continuous state variables and altering at least one of the tunable drift terms.

22. The system of either claim 2 or claim 20, wherein the processor comprises analog circuitry configured to evolve over time simultaneously with the physical system.

23. The system of claim 22, wherein the analog circuitry is configured to continuously output predictions for score values as analog signals to be used as inputs to the physical system.

24. The system of claim 23, wherein the score values are produced by physically modeling, with trainable physical devices, partial derivatives of the score values with respect to time and with respect to the continuous state variables.

25. The system of claim 24, wherein the trainable physical devices used to model the partial derivatives are artificial neural networks.

26. The system of claim 24, further comprising:

an integrator circuit, operably coupled to the trainable physical devices, to produce the score values by integrating, over time, outputs of the trainable physical devices.

27. The system of claim 22, wherein the processor further comprises a field-programmable gate array (FPGA).

28. The system of claim 22, wherein the analog circuitry comprises a network of electrical circuits, each electrical circuit in the network of electrical circuits providing a corresponding degree of freedom and comprising:

a capacitor;

a resistor in series with the capacitor; and

a voltage source in series with the capacitor.

29. The system of claim 28, wherein at least one electrical circuit in the network of electrical circuits is capacitively coupled to at least one other electrical circuit in the network of electrical circuits according to a problem geometry associated with the dataset.

30. The system of claim 28, wherein the voltage source is a multi-layer neural network configured to take the continuous state variable as input and to output a voltage value for each electrical circuit in the network of electrical circuits.

31. The system of claim 1, wherein the physical system comprises switches configured to be actuated between settings for a forward diffusion process and settings for a reverse diffusion process.

32. A system for executing a Bayesian neural network, the system comprising:

a physical system having a plurality of degrees of freedom, each degree of freedom in the plurality of degrees of freedom associated with a corresponding weight in the Bayesian neural network and having a continuous state variable that evolves according to a corresponding differential equation having a diffusion term and a drift term.

33. The system of claim 32, wherein the physical system comprises a network of electrical circuits, each electrical circuit in the network of electrical circuits providing a corresponding degree of freedom.

34. The system of claim 33, wherein each electrical circuit in the network of electrical circuits comprises:

a capacitor having a charge or voltage encoding the continuous state variable;

a stochastic noise source, in series with the capacitor, to generate the diffusion term; and

a resistor, in series with the capacitor, to generate the drift term.

35. The system of claim 34, wherein the stochastic noise source is a thermal noise source and/or a shot noise source.

36. The system of claim 34, wherein each electrical circuit in the network of electrical circuits further comprises:

an amplifier, operably coupled to the stochastic noise source, to amplify an output of the stochastic noise source so as to tune the diffusion term of the corresponding differential equation.

37. The system of claim 33, further comprising:

s witches coupling the electrical circuits in the network of electrical circuits according to a problem geometry associated with the Bayesian neural network.

38. The system of claim 32, further comprising:

a processor, operably coupled to the physical system, to modify dynamics of continuous state variables of the differential equations to produce a posterior distribution for the weights of the Bayesian neural network.

39. The system of claim 38, wherein the processor is configured to read off the continuous state variables from the physical system and to alter at least one of the drift terms.

40. The system of claim 38, wherein the processor is configured to be trained by an optimization routine that reduces a loss function.

41. The system of claim 40, wherein the physical system is configured to assist with estimating the loss function.

42. The system of claim 40, further comprising:

analog circuits, operably coupled to the processor, to assist in computing time integrals for estimating the loss function.

43. The system of claim 40, wherein the physical system is configured to assist in estimating gradients of the loss function.

44. The system of claim 38, wherein the processor comprises a field-programmable gate array (FPGA).

45. The system of claim 38, wherein the processor comprises analog circuitry configured to evolve over time simultaneously with the physical system.

46. The system of claim 45, wherein the analog circuitry is configured to continuously output predictions for the drift terms as analog signals to be used as inputs to the physical system.

47. The system of claim 46, wherein the predictions are produced by physically modeling, with trainable physical devices, partial derivatives of the drift terms with respect to time and with respect to the continuous state variables.

48. The system of claim 47, wherein the trainable physical devices used to model the partial derivatives

implement artificial neural networks.

49. The system of claim 47, further comprising:
at least one integrator circuit, operably coupled to the trainable physical devices, to produce the drift terms by integrating, over time, outputs of the trainable physical devices.

50. The system of claim 32, further comprising:
a processor to implement a model whose weights correspond to continuous state variables of the physical system.

51. The system of claim 50, wherein the processor comprises a field-programmable gate array (FPGA).

52. The system of claim 50, wherein the processor comprises a network of electrical circuits, each electrical circuit in the network of electrical circuits providing a corresponding latent variable of the model.

53. The system of claim 52, wherein each electrical circuit in the network of electrical circuits comprises:
a capacitor having a charge or voltage encoding the corresponding latent variable; and
a non-linear element to provide a non-linear activation function of the model.

54. The system of claim 53, wherein the non-linear element comprises at least one of a transistor or a diode.

55. The system of claim 52, wherein the network of electrical circuits comprises:
variable resistors having resistances controlled by external voltage sources.

56. The system of claim 52, wherein the network of electrical circuits comprises a number of unit cells greater than or equal to a dimension of a feature space of an input to the Bayesian neural network.

57. A system for executing a neural ordinary differential equation (ODE), the system comprising:
a physical system having a plurality of degrees of freedom, each degree of freedom associated with a hidden layer unit of the neural ODE and having a corresponding continuous state variable that evolves according to a differential equation having a non-linear term and an affine term.

58. The system of claim 57, wherein the physical system comprises a network of electrical circuits, each electrical circuit in the network of electrical circuits providing a corresponding degree of freedom.

59. The system of claim 58, wherein each electrical circuit in the network of electrical circuits comprises:
a capacitor having a charge or voltage encoding the corresponding continuous state variable; and
a non-linear element to generate the corresponding non-linear term.

60. The system of claim 58, wherein the network of electrical circuits comprises variable resistors to generate the affine terms, the variable resistors having trainable resistances.

61. The system of claim 57, wherein the physical system has revesable dynamics.

62. The system of claim 61, further comprising:
an additional physical system to evolve an adjoint variable simultaneously with evolution of the continuous state variables.

63. The system of claim 57, wherein the physical system is configured to perform time-series data fitting or extrapolation.

64. The system of claim 58, wherein each electrical circuit in the network of electrical circuits includes a corresponding stochastic noise source.

65. A processor configured to implement a Bayesian neural network, the processor comprising:
an analog weight diffuser having stochastic dynamics that generate a trajectory of weight values for the Bayesian neural network;
a hidden layer network, operably coupled to analog weight diffuser, to generate an output vector based on an input vector and the trajectory of weight values; and
a posterior drift network, operably coupled to the analog weight diffuser and the hidden layer network, to guide the trajectory of weight values based on the output vector.

66. The processor of claim 65, wherein the trajectory of weight values is a solution of a stochastic differential equation representing the stochastic dynamics of the analog weight diffuser.

67. The processor of claim 66, wherein the analog weight diffuser comprises a plurality of interconnected unit cells, each interconnected unit in the plurality of interconnected unit cells comprising:
a resistor to represent a drift term of the stochastic differential equation;
a stochastic noise source, in series with the resistor, source to generate a diffusion term of the stochastic differential equation; and
a capacitor, in series with the resistor and the stochastic noise source, to encode a continuous state variable that evolves according to the stochastic differential equation.

68. The processor of claim 67, wherein the stochastic noise source comprises at least one of a shot noise source or a thermal noise source.

69. The processor of claim 65, wherein the analog weight diffuser is configured to be switched between a first configuration in which the analog weight diffuser generates a posterior weight distribution and a second configuration in which the analog weight diffuser generates a prior weight distribution.

70. The processor of claim 65, wherein the hidden layer network comprises at least one of a processing unit or a field-programmable gate array.

71. The processor of claim 65, wherein the hidden layer network comprises a network of analog unit cells.

72. The processor of claim 71, wherein each analog unit cell in the network of analog unit cells comprises:
a capacitor to encode a continuous state variable that evolves according to a neural ordinary differential equation (ODE);
a resistor, in series with the capacitor, to provide a time derivative of the continuous state variable;
a voltage source, in parallel with the capacitor, to at least partially provide a linear drift term for

the continuous state variable; and

a nonlinear element, in parallel with the capacitor, to provide a nonlinear drift term for the continuous state variable.

73. The processor of claim 72, wherein the nonlinear element comprises at least one of a transistor or a diode.

74. The processor of claim 71, wherein analog unit cells in the network of analog unit cells are connected by switches and resistive bridges in a geometry based on the input vector.

75. The processor of claim 65, wherein the posterior drift network comprises at least one of a processing unit or a field-programmable gate array, and further comprising:

an analog-to-digital converter, operably coupling the analog weight diffuser to the posterior drift network, to convert an analog output from the analog weight diffuser into a digital input to the at least one of the processing unit or the field-programmable gate array; and

a digital-to-analog converter, operably coupling the posterior drift network to the analog weight diffuser, to convert a digital output from the at least one of the processing unit or the field-programmable gate array into an analog input to the analog weight diffuser.

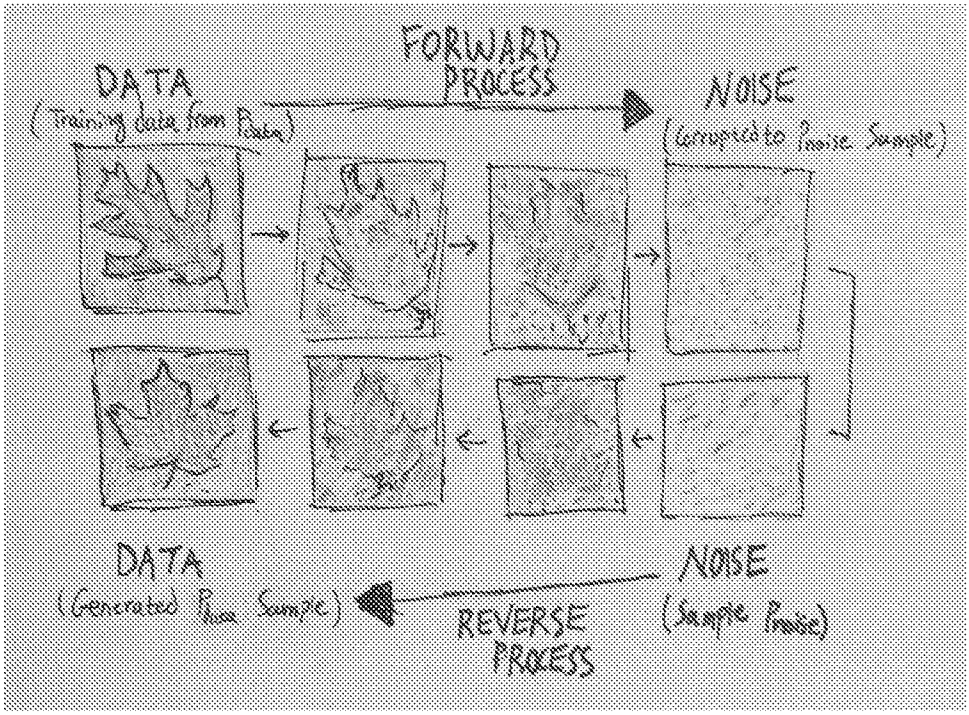

Figure 1

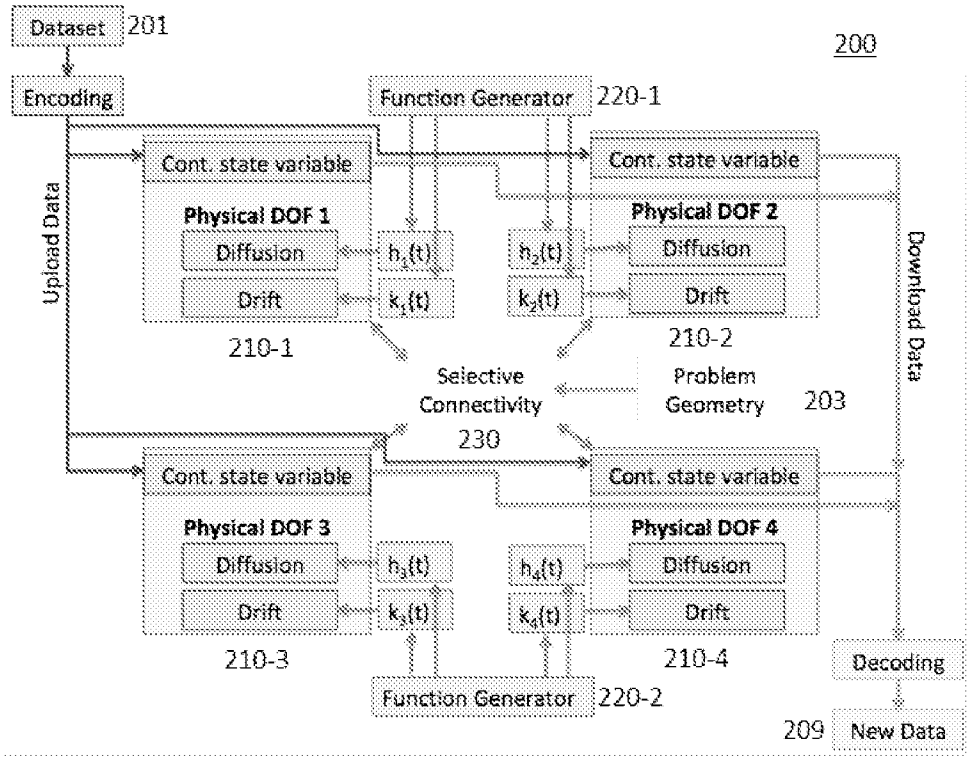

Figure 2

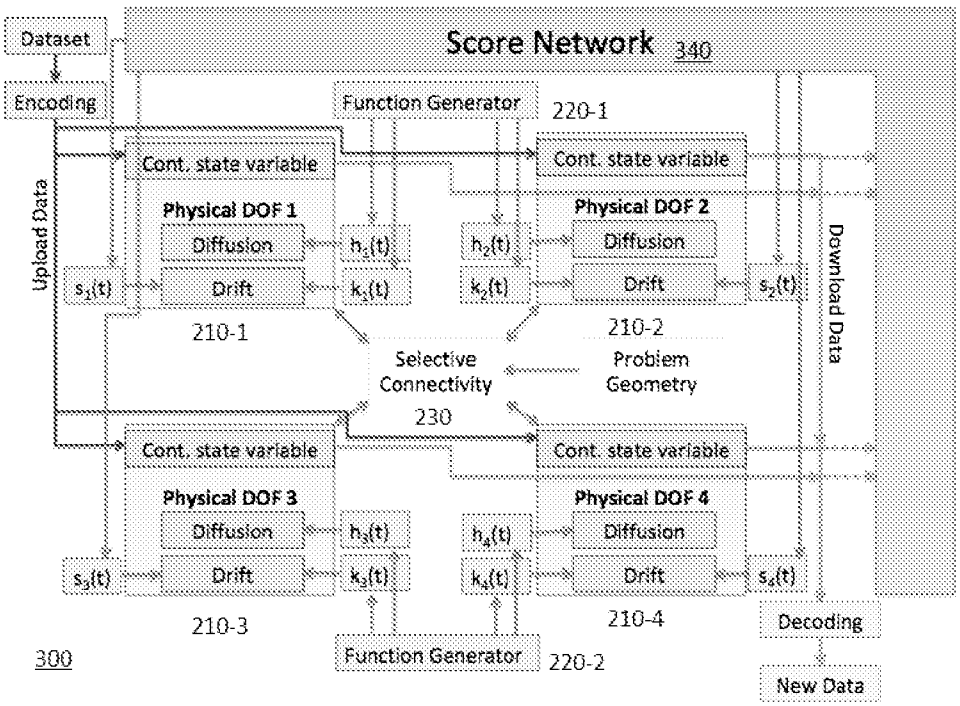

Figure 3

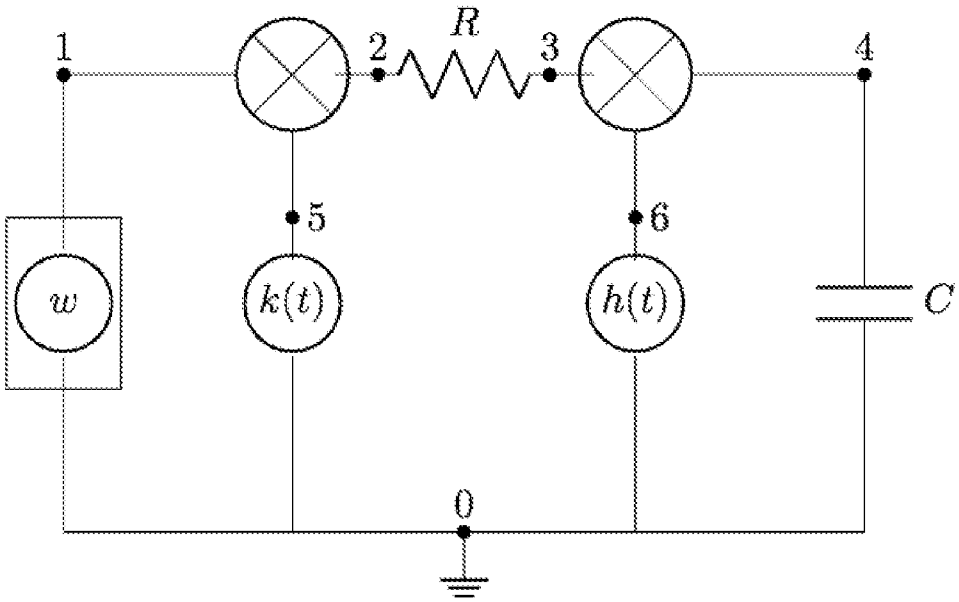

Figure 4

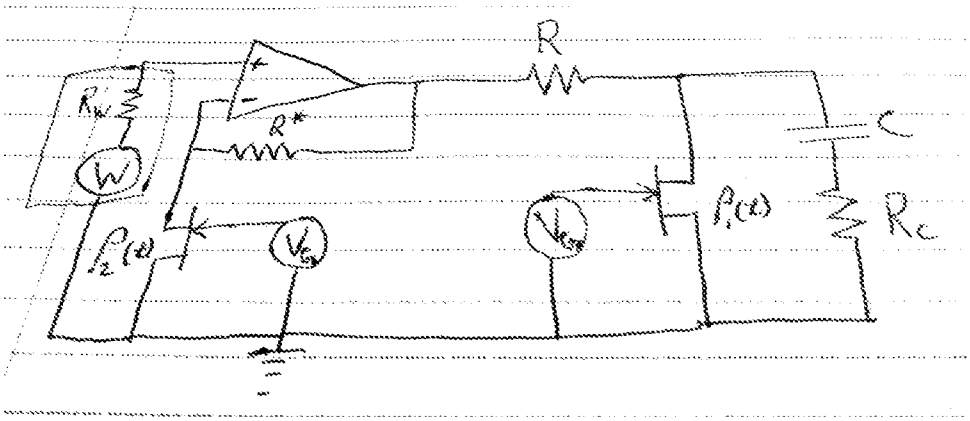

Figure 5

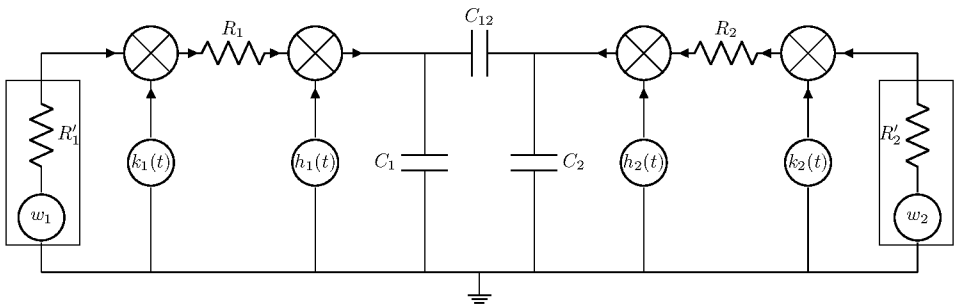

Figure 6

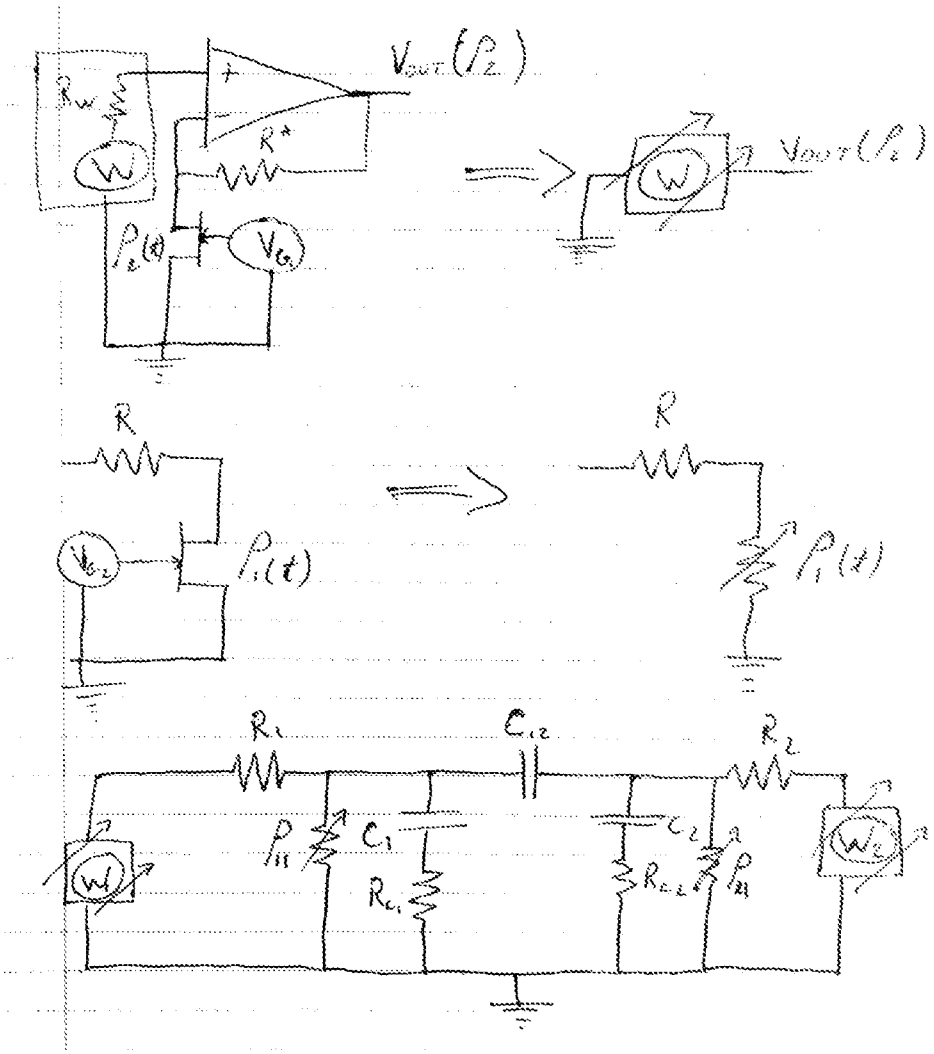

Figure 7

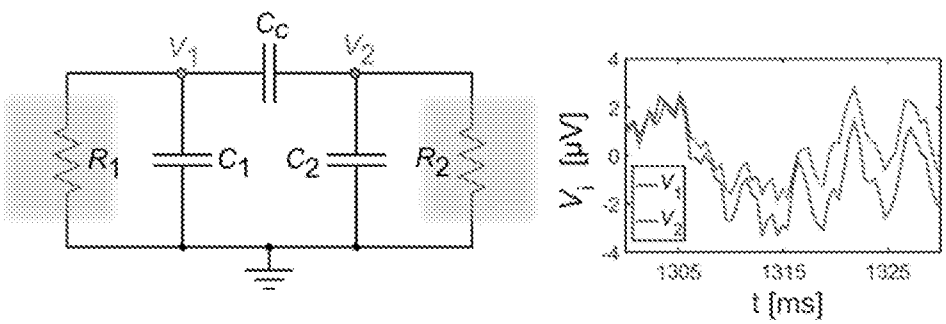

Figure 8

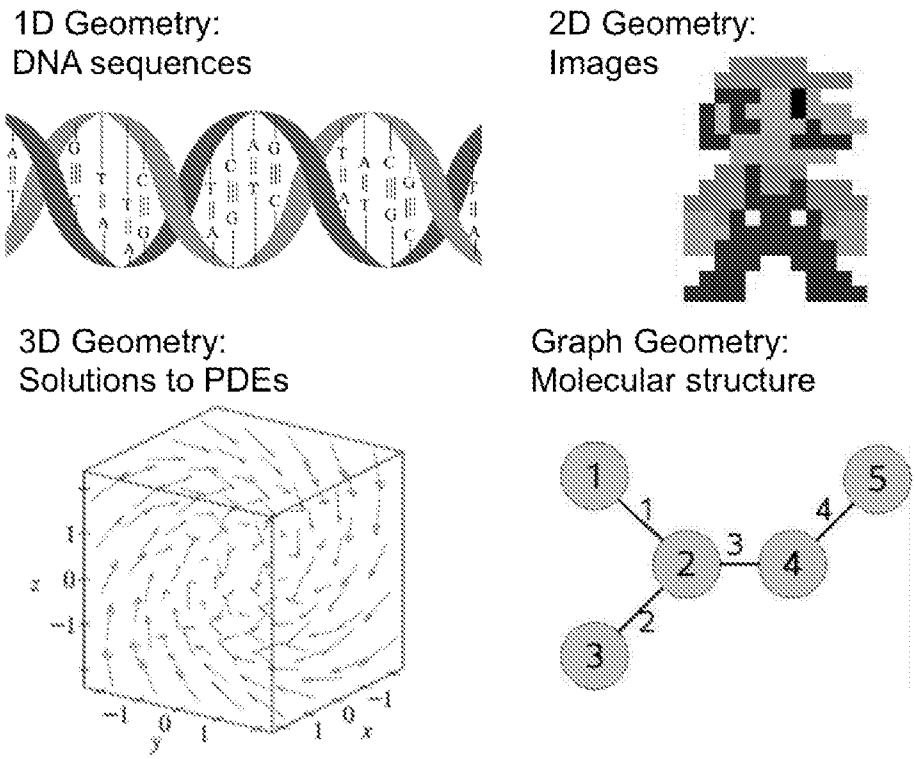

Figure 9

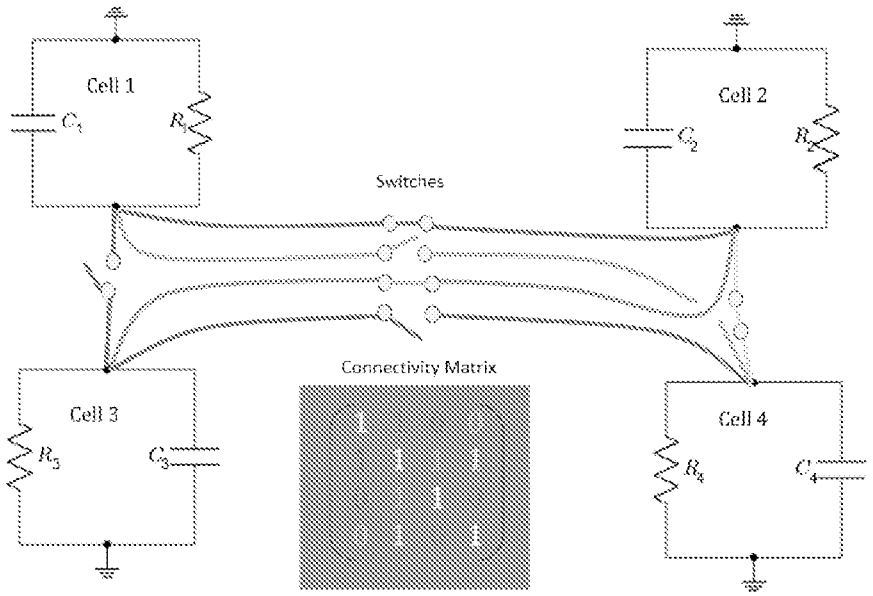

Figure 10

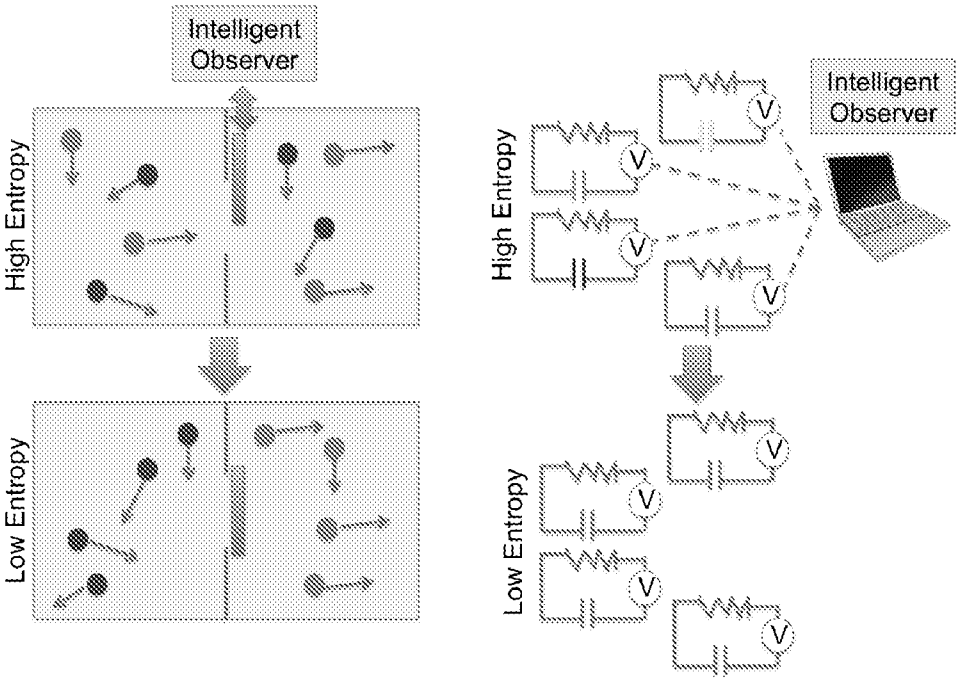

Figure 11

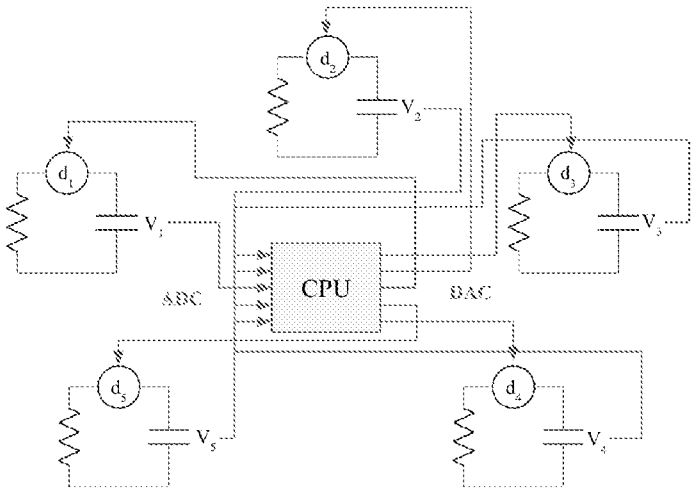

Figure 12

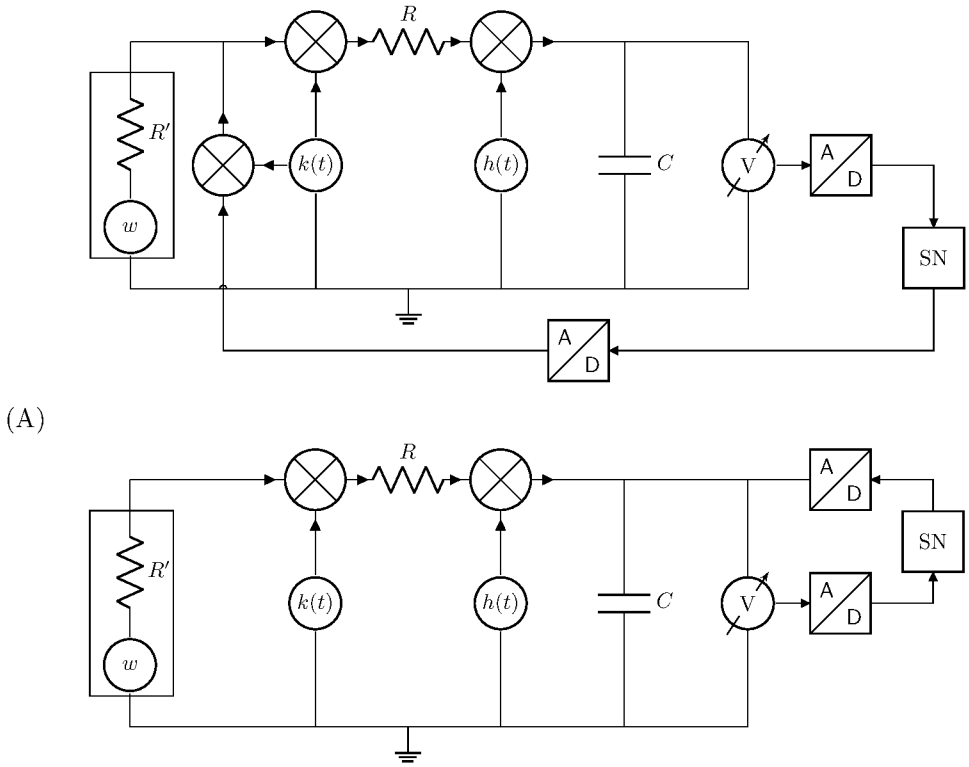

Figure 13

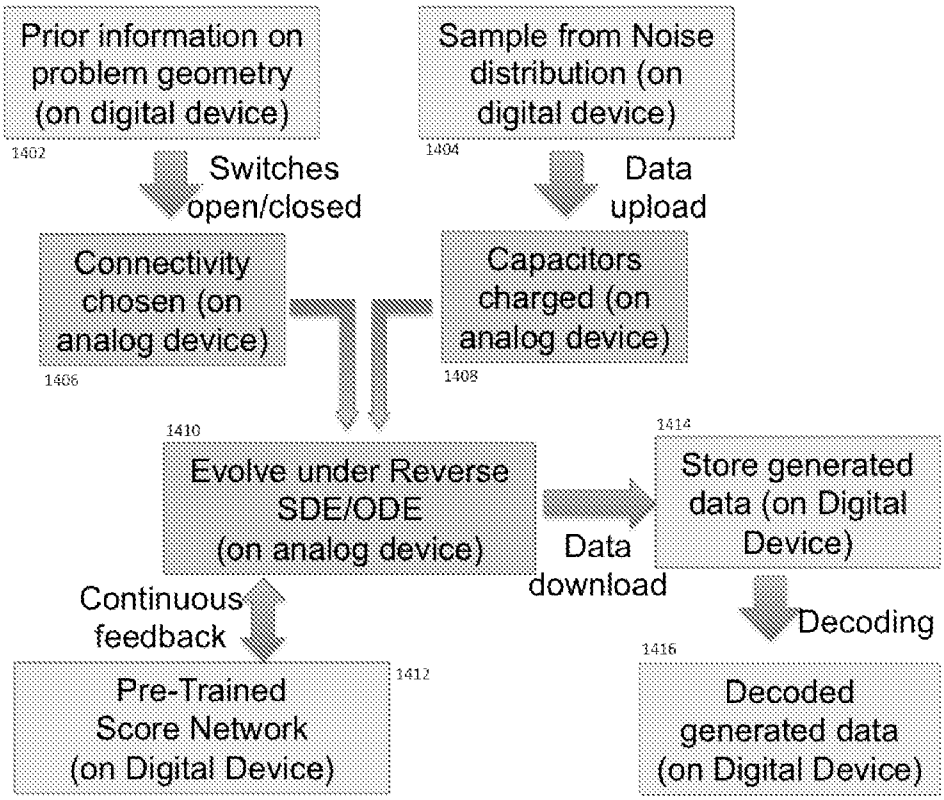

Figure 14

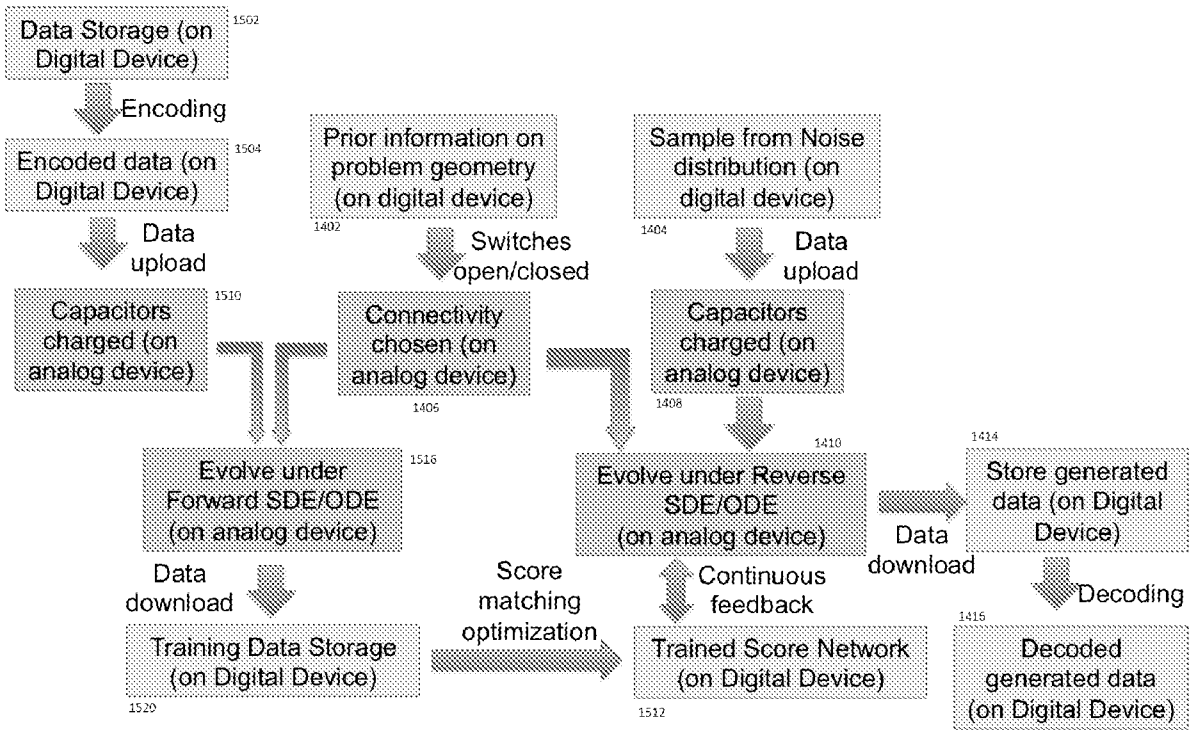

Figure 15

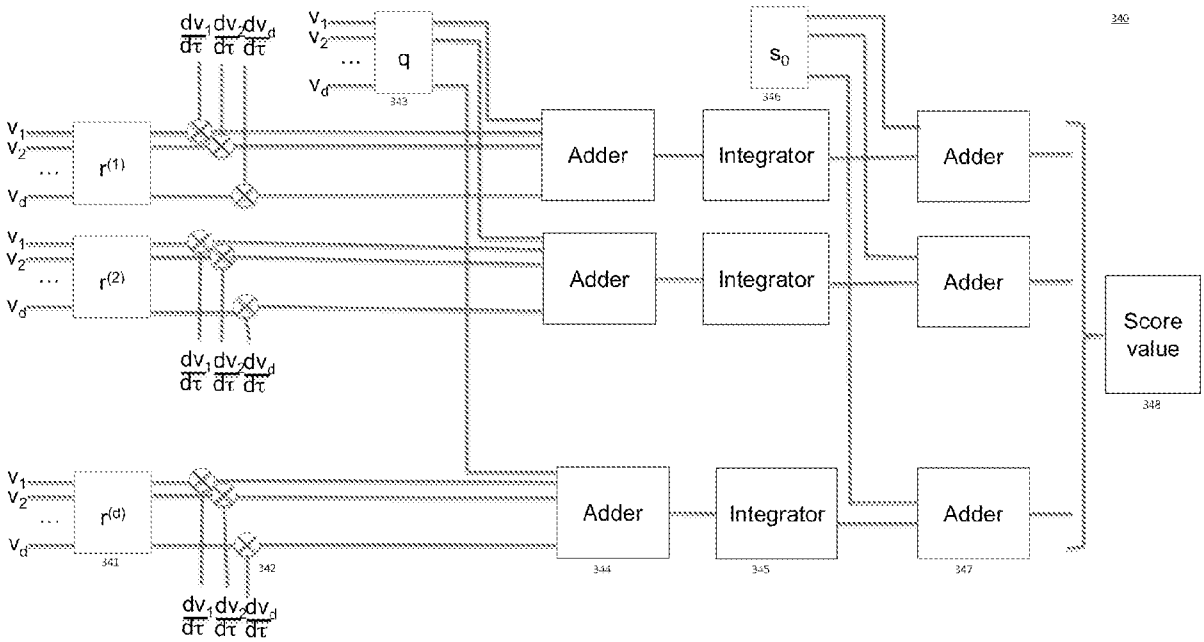

Figure 16

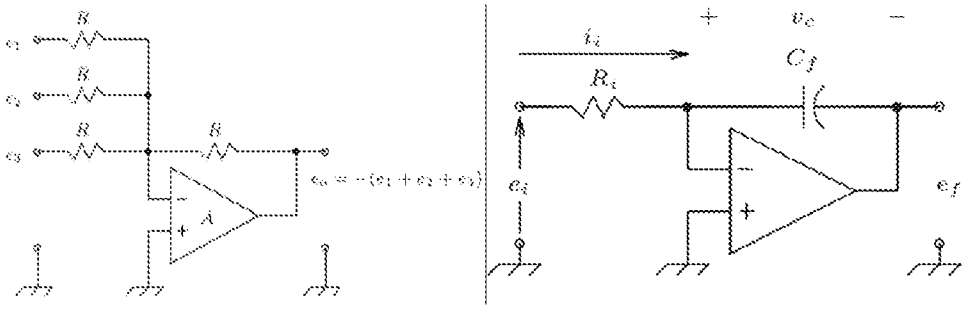

Figure 17

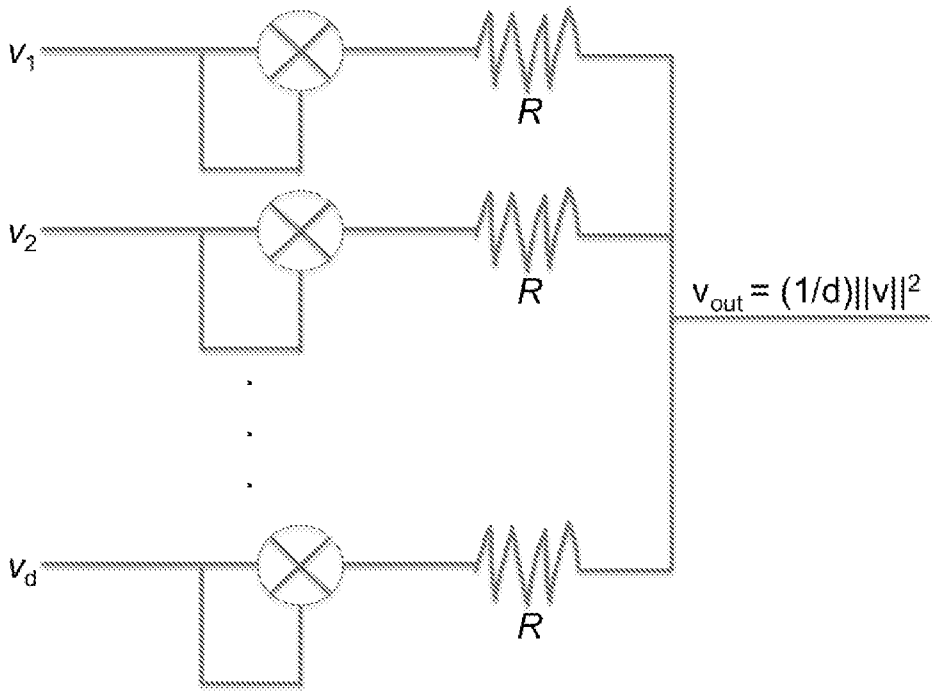

Figure 18

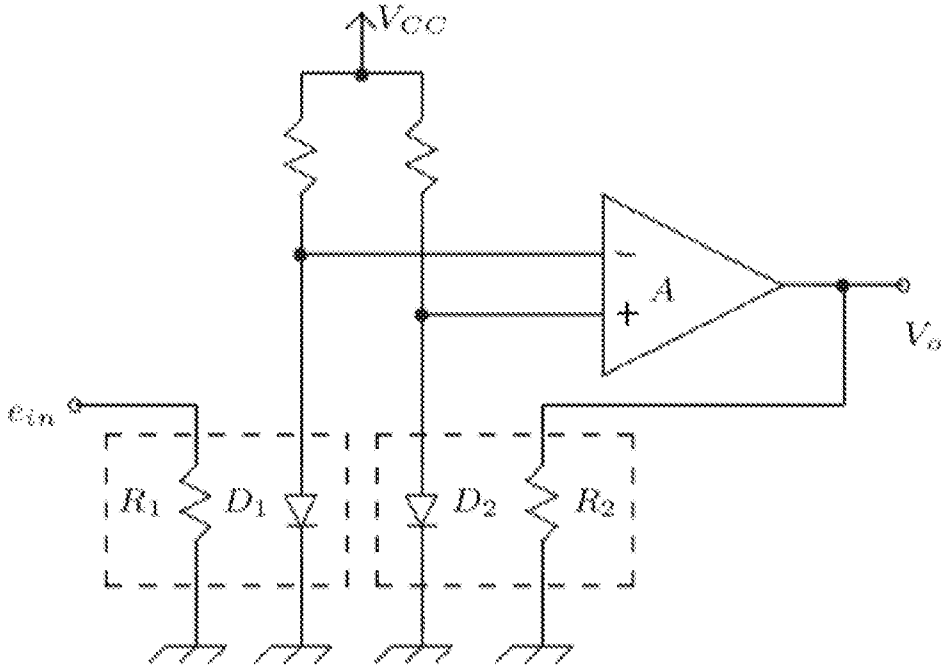

Figure 19

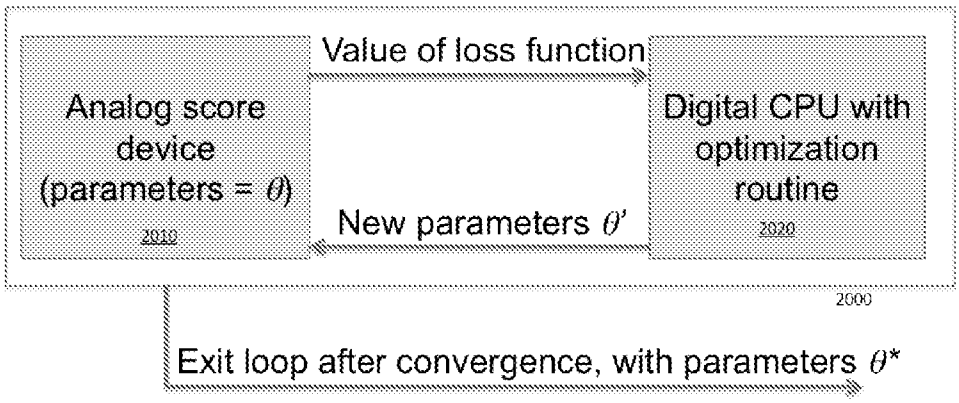

Figure 20

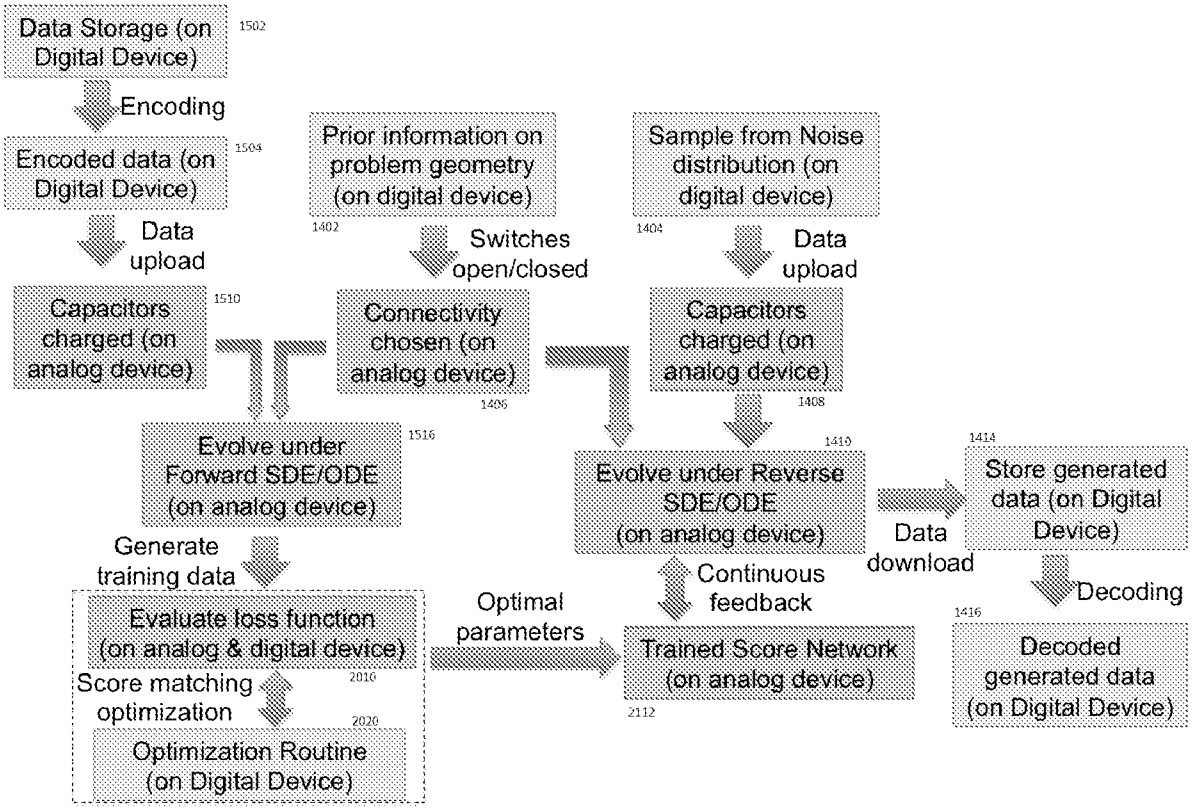

Figure 21

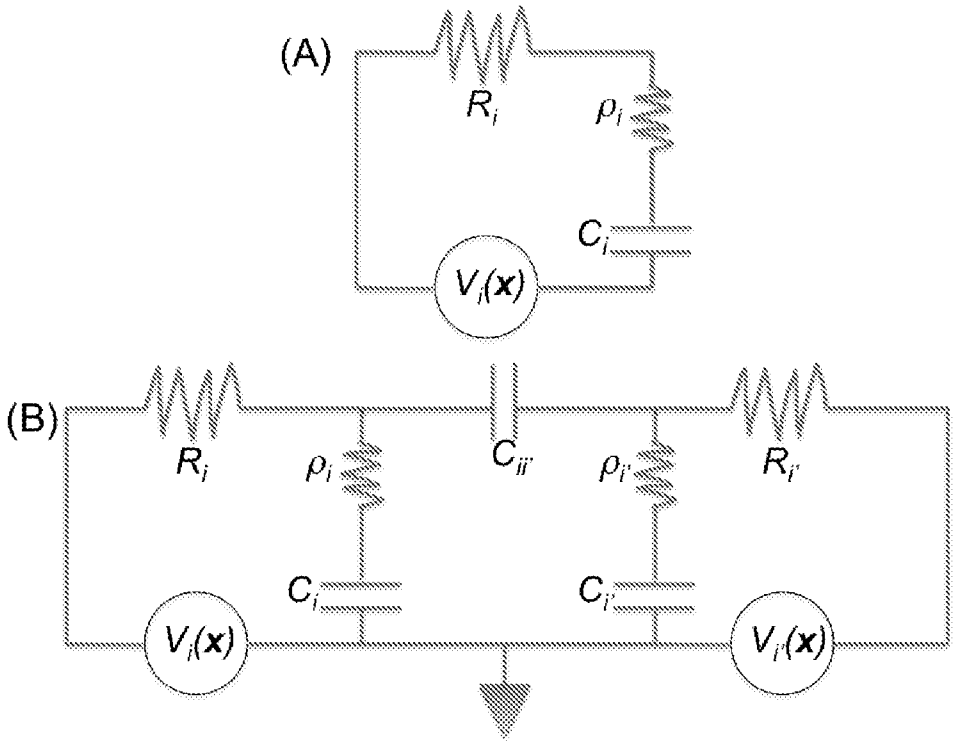

Figure 22

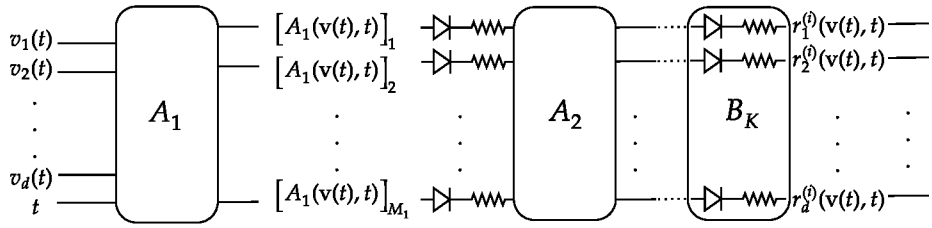

Figure 23

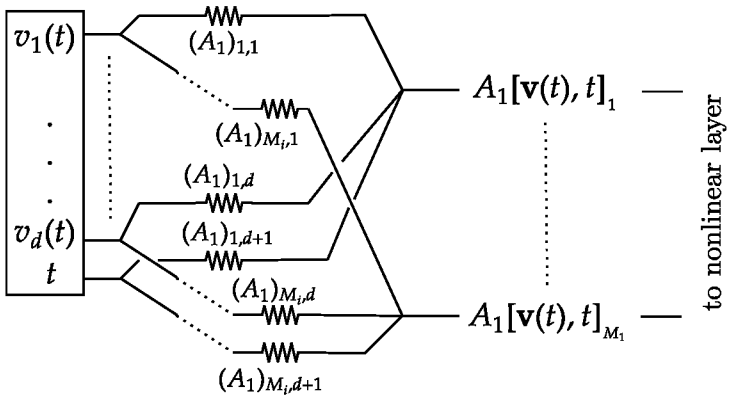

Figure 24

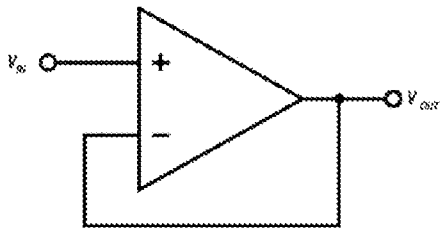

Figure 25

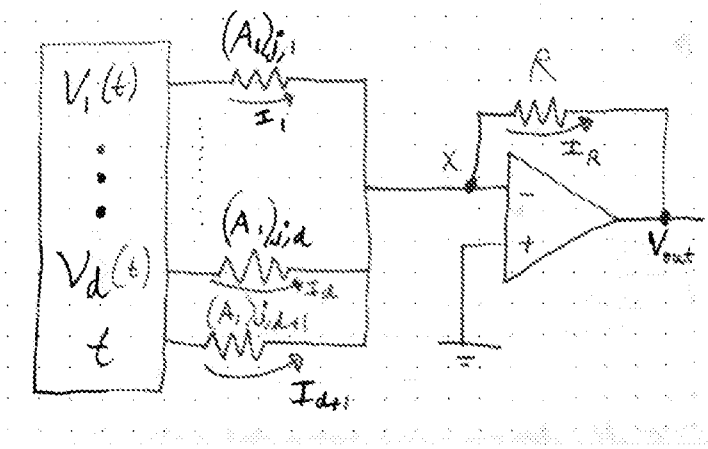

Figure 26

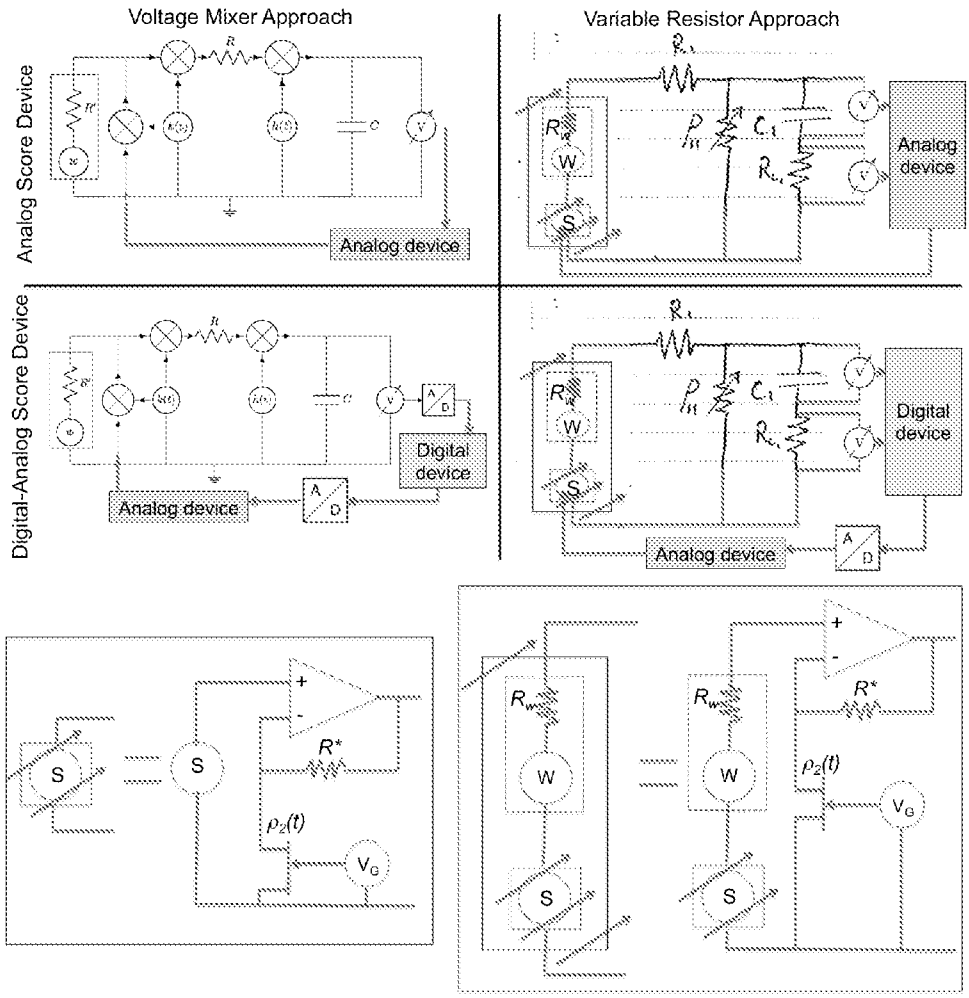

Figure 27

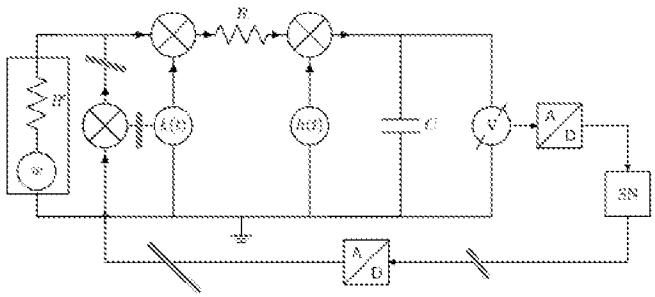

Figure 28

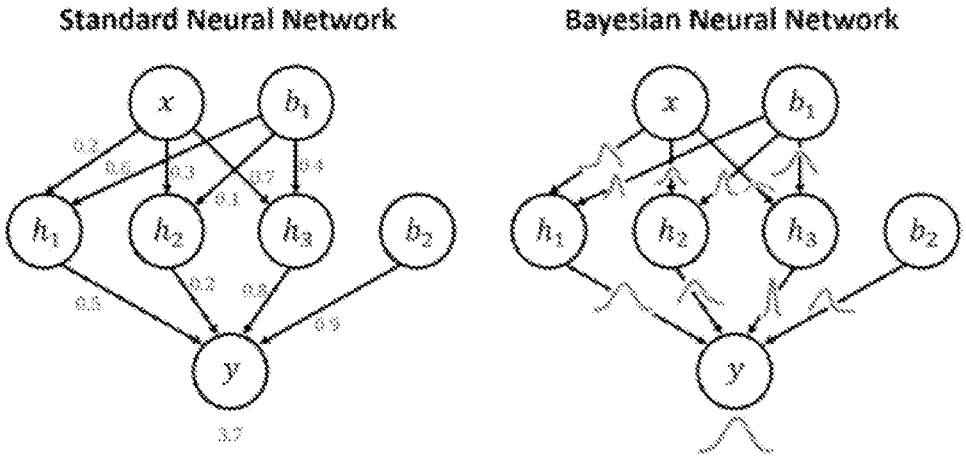

Figure 29

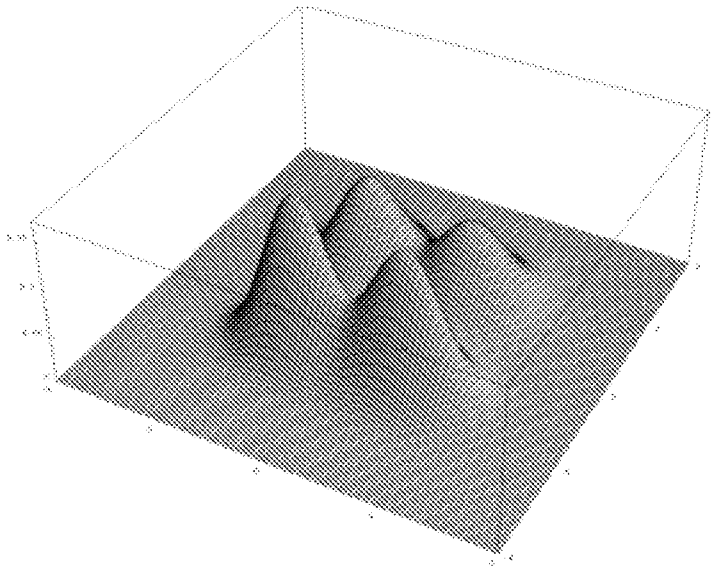

Figure 30

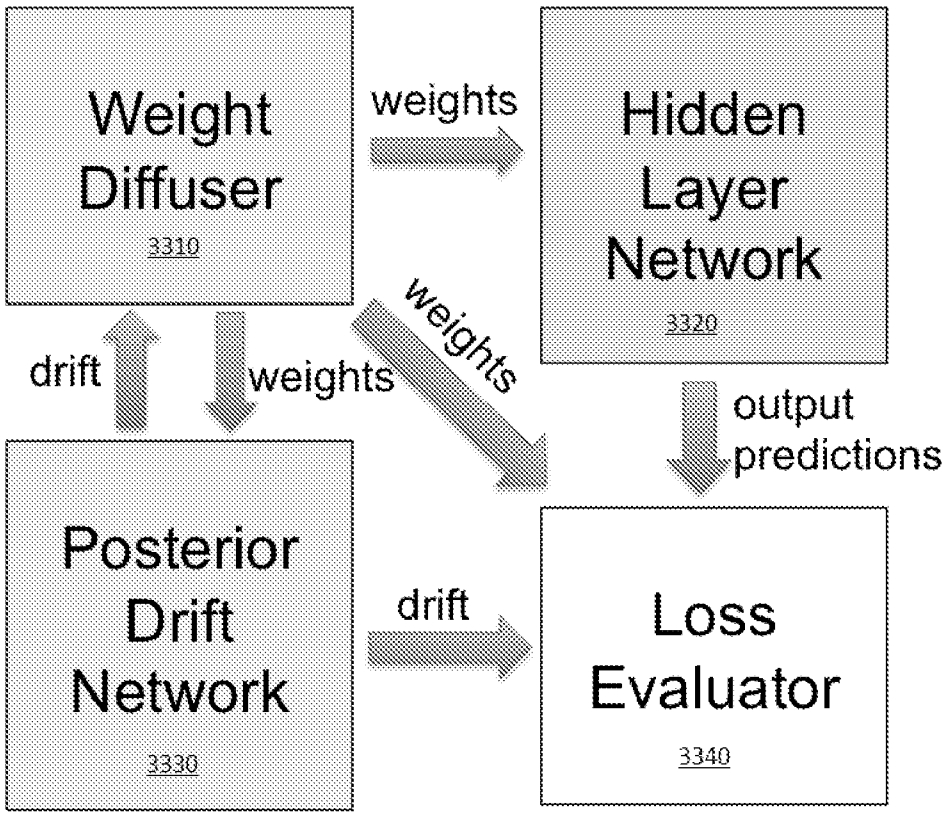

Figure 31

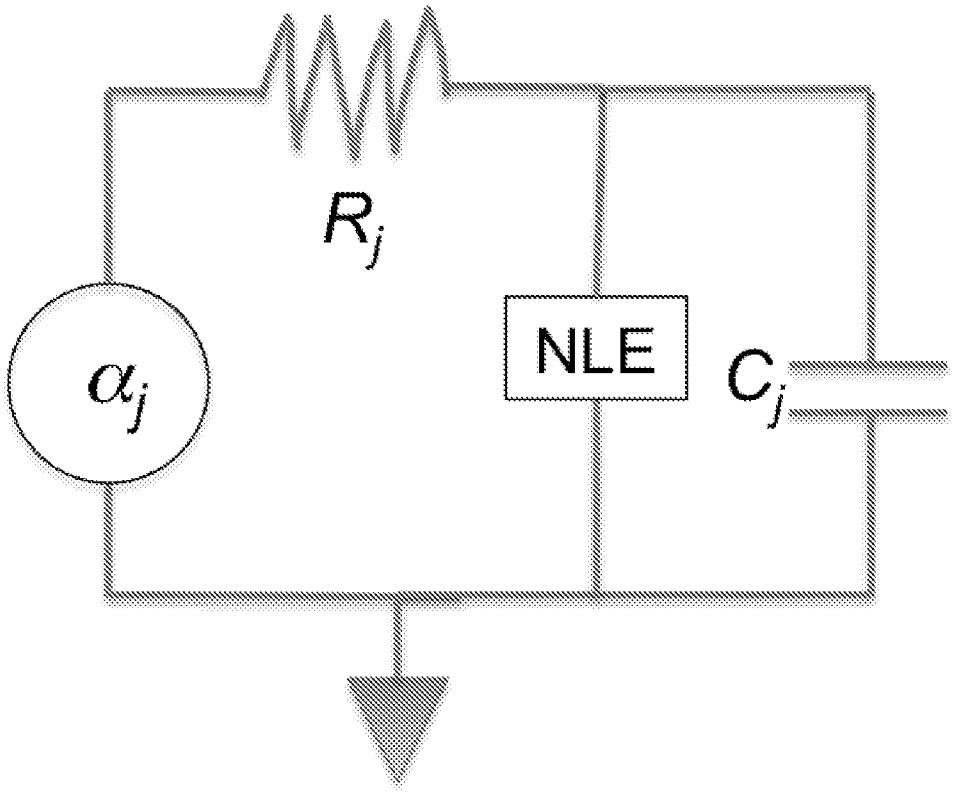

Figure 32

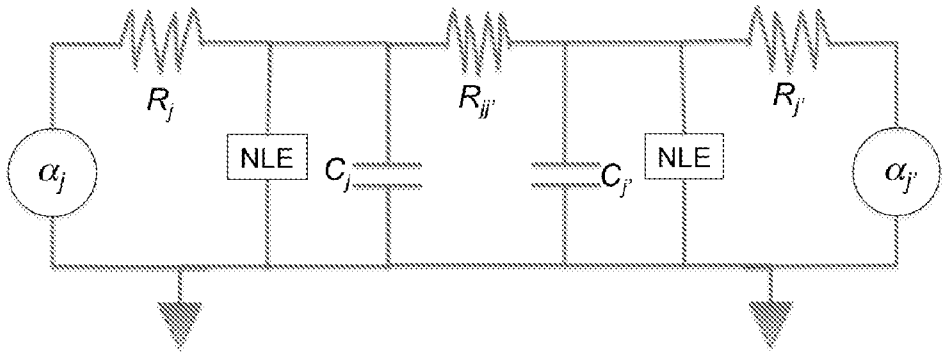

Figure 33

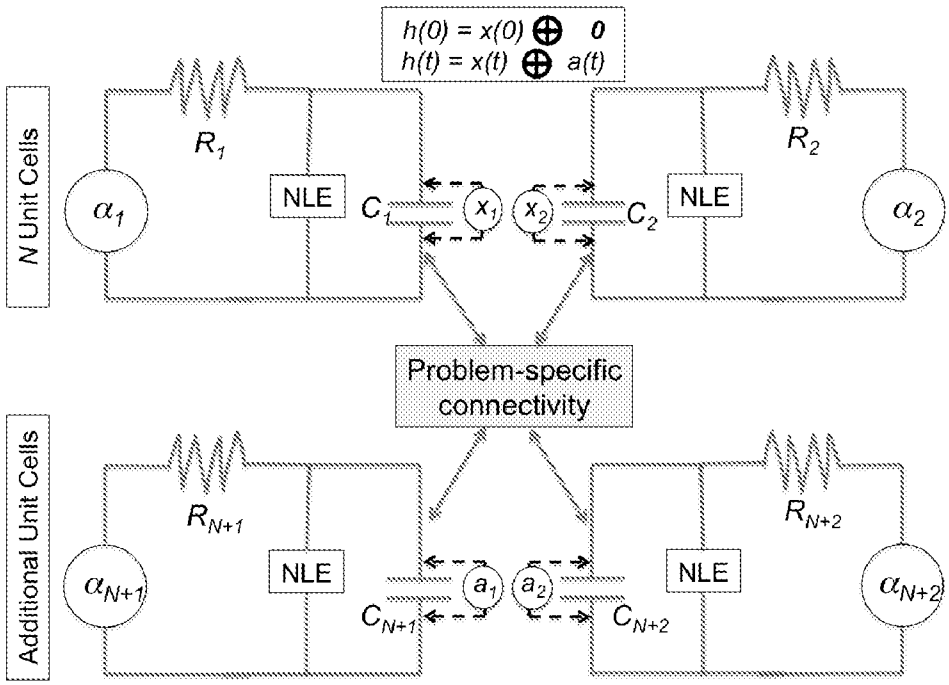

Figure 34

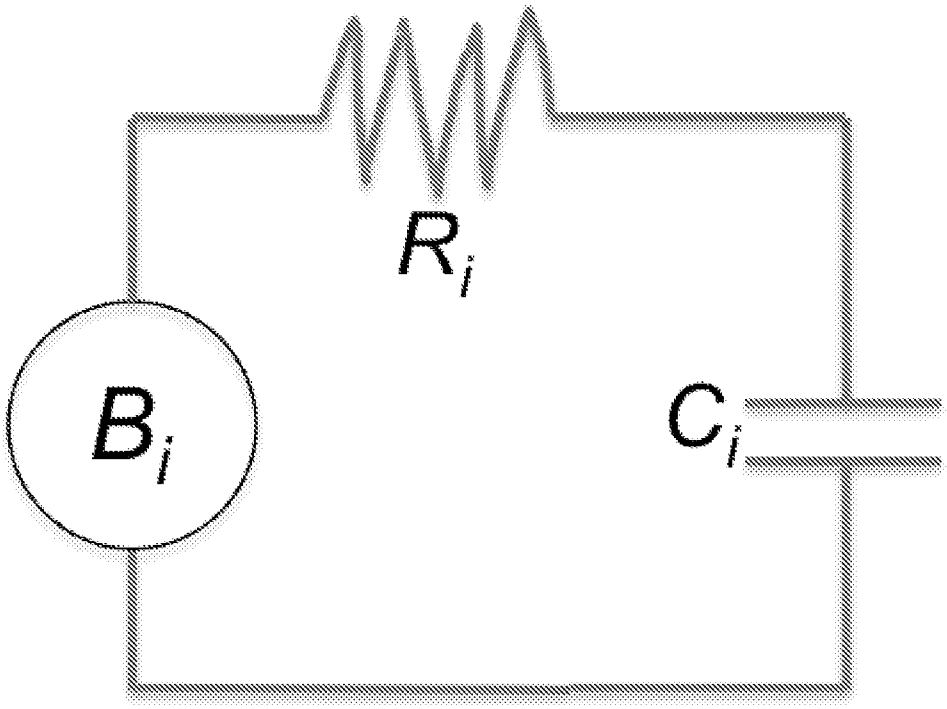

Figure 35

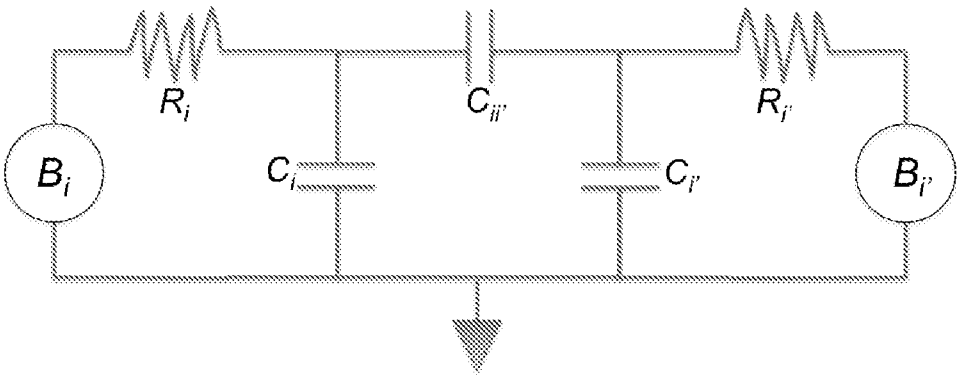

Figure 36

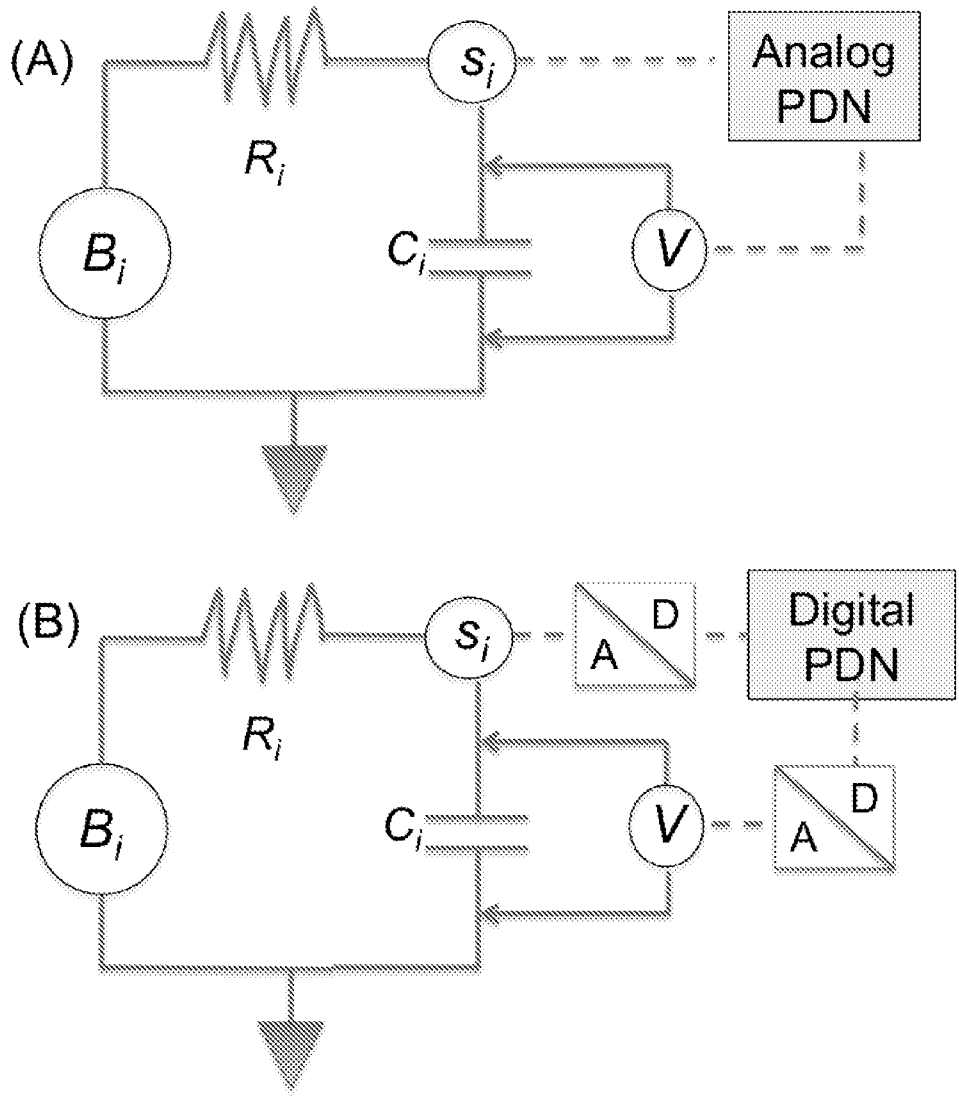

Figure 37

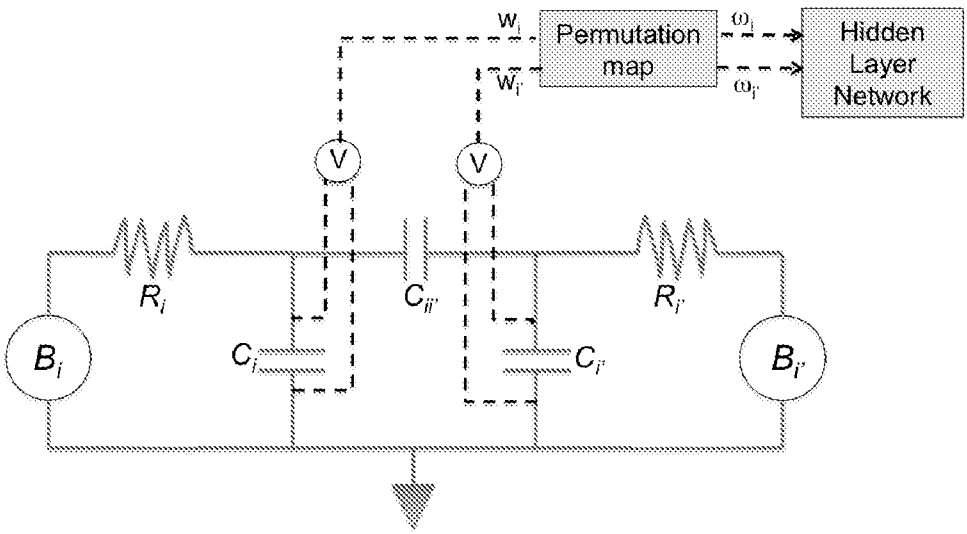

Figure 38

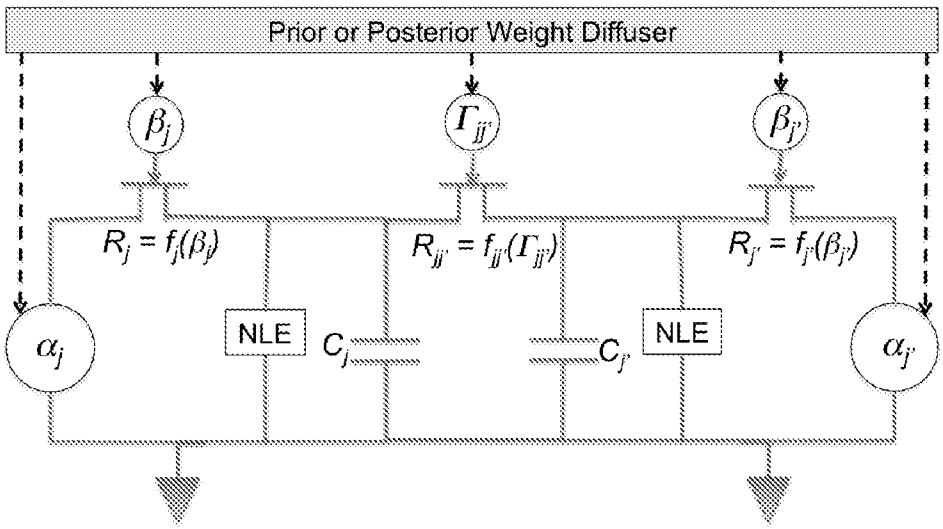

Figure 39

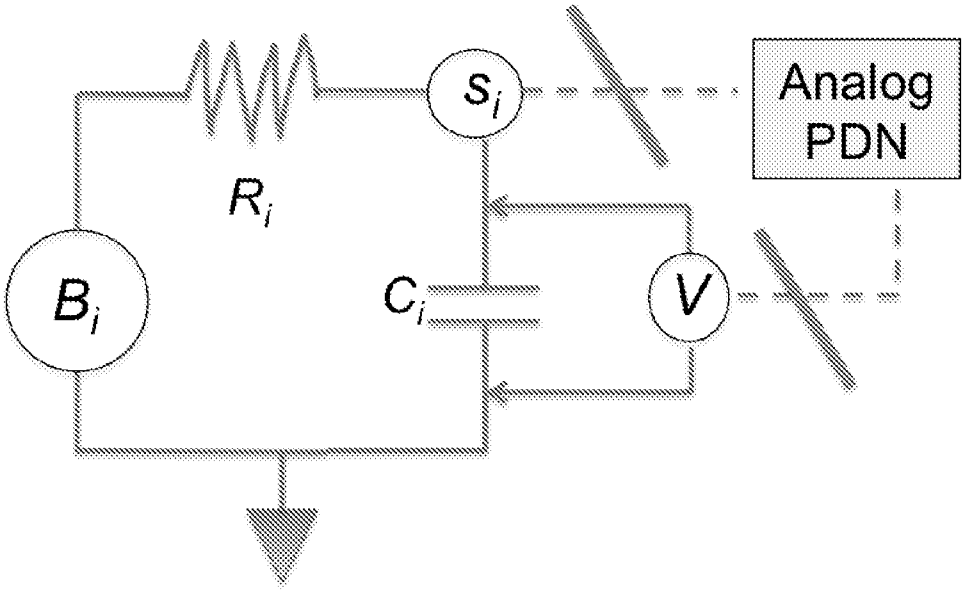

Figure 40

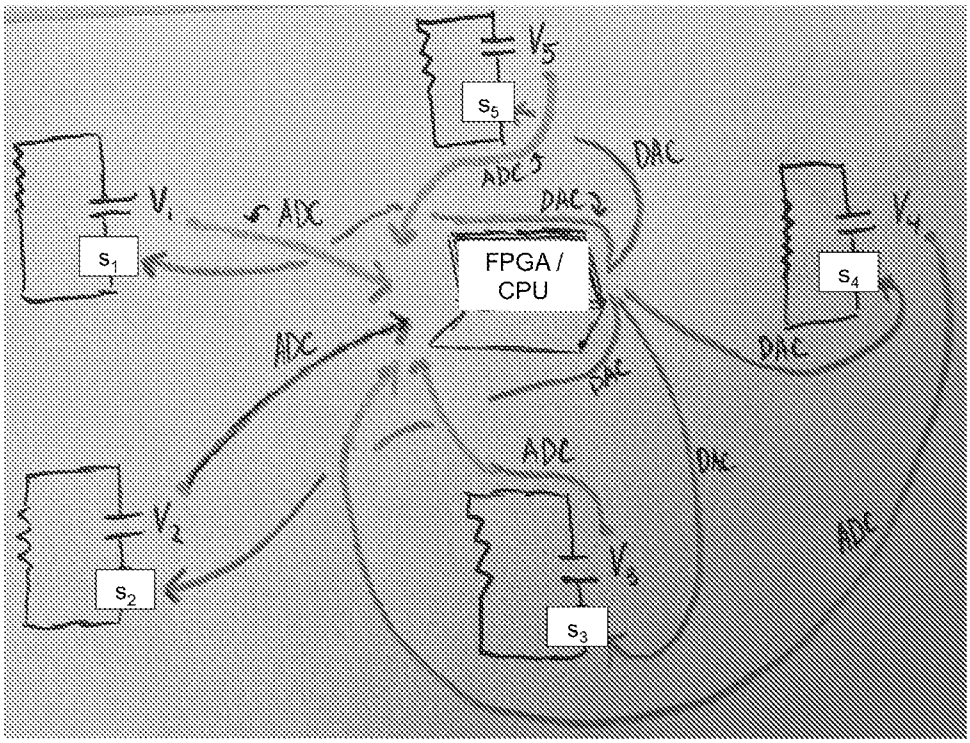

Figure 41

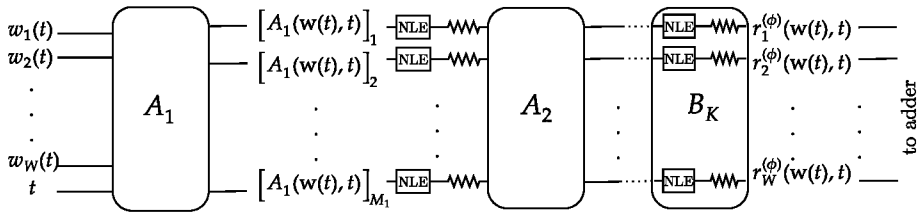

Figure 42

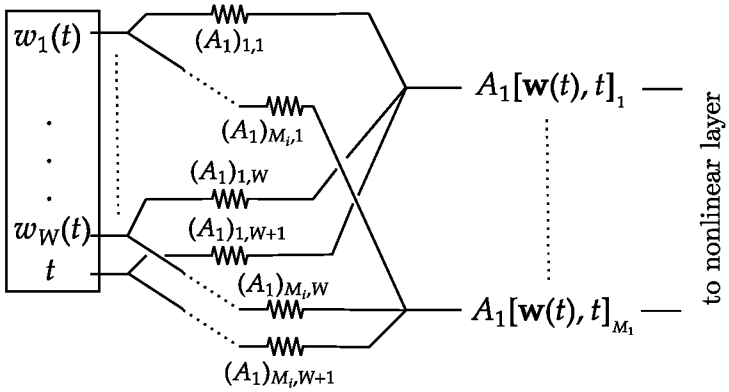

Figure 43

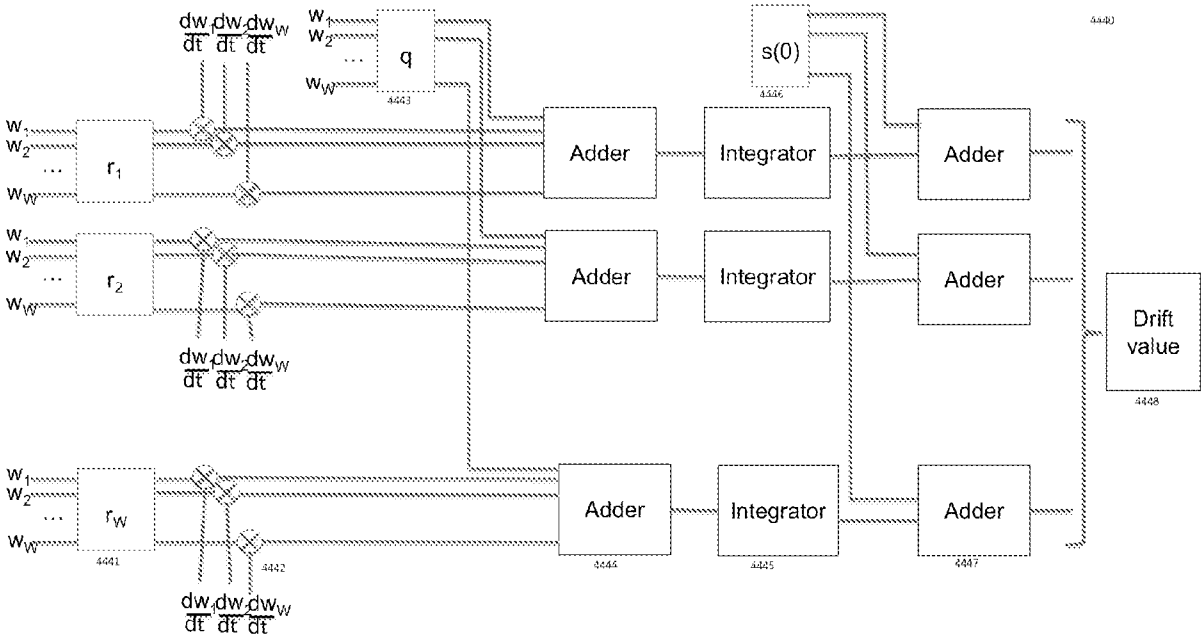

Figure 44

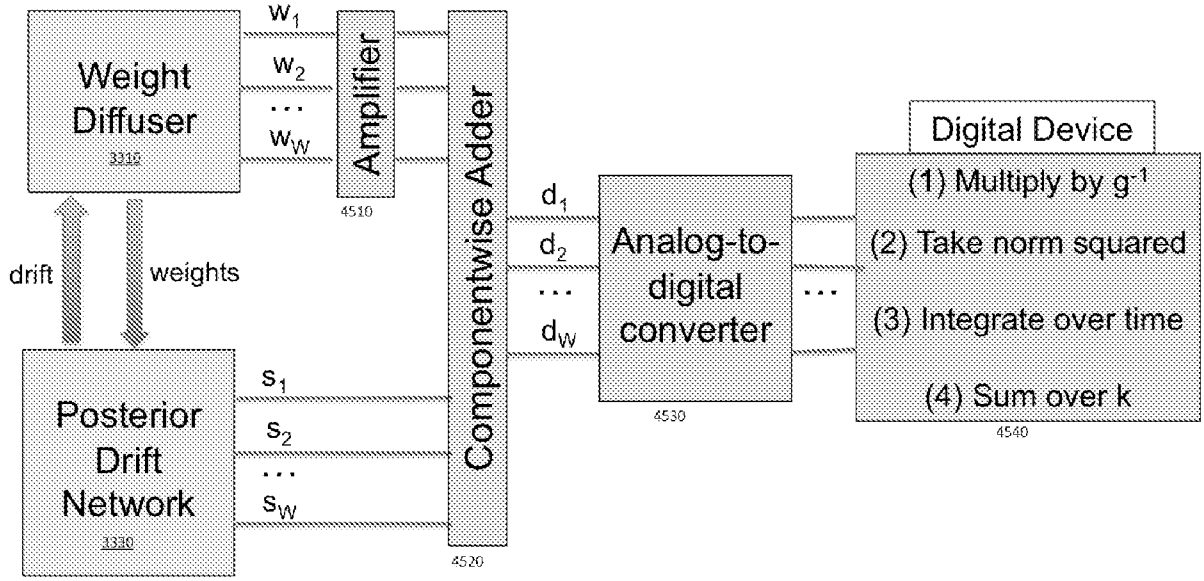

Figure 45

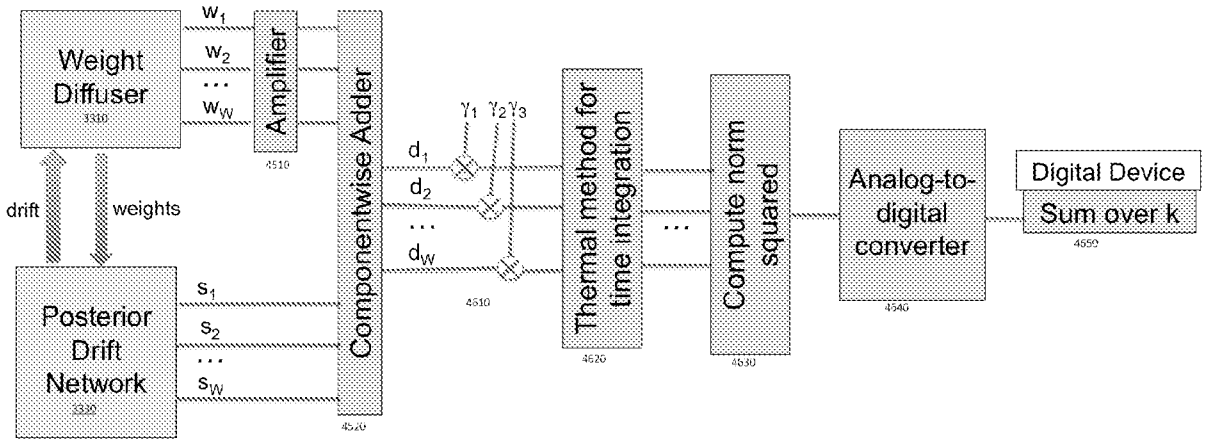

Figure 46

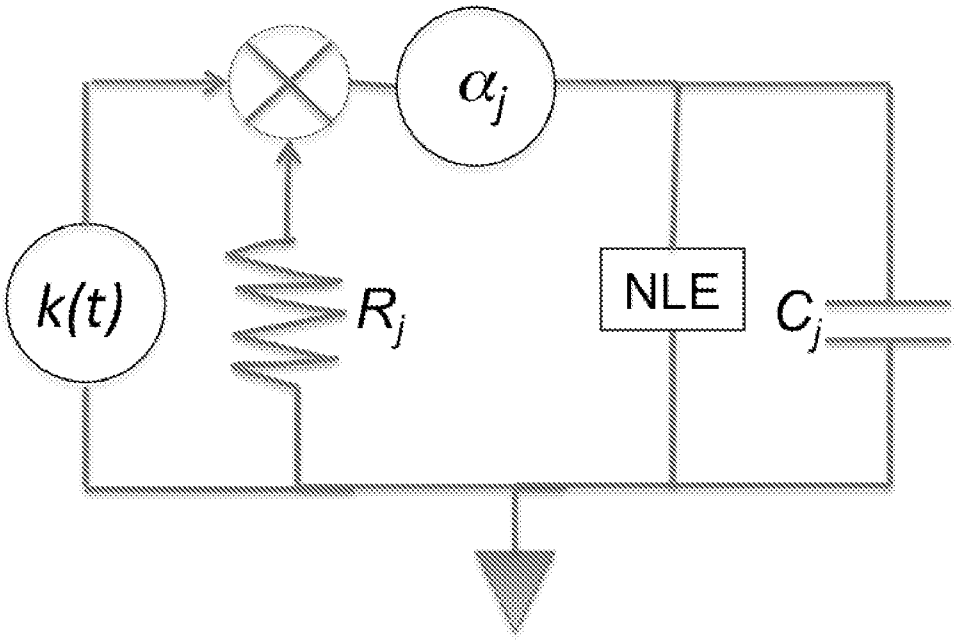

Figure 47

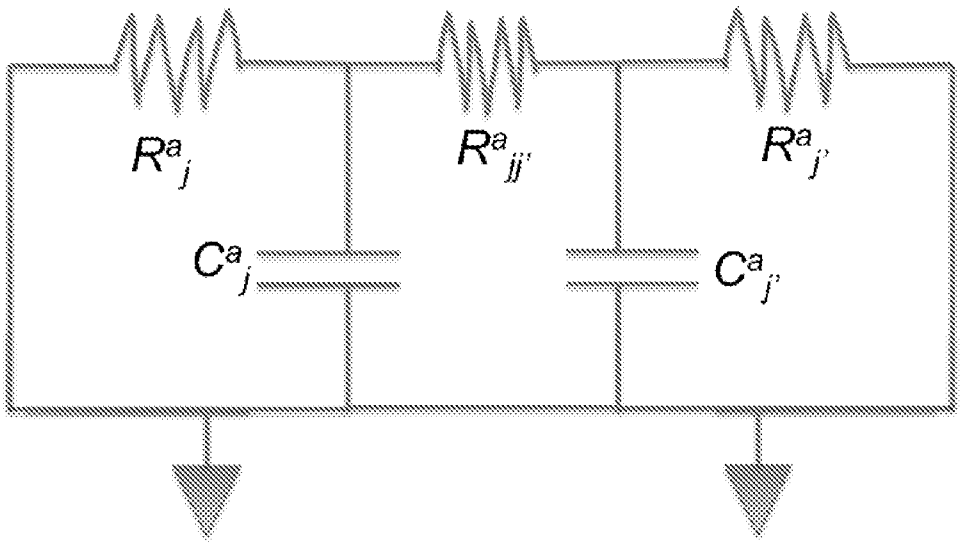

Figure 48

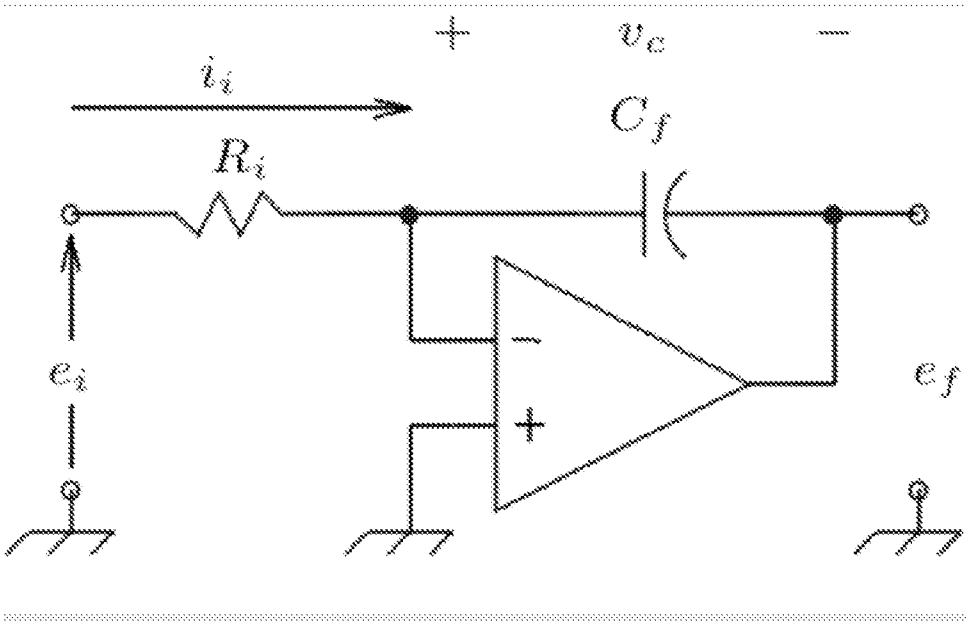

Figure 49

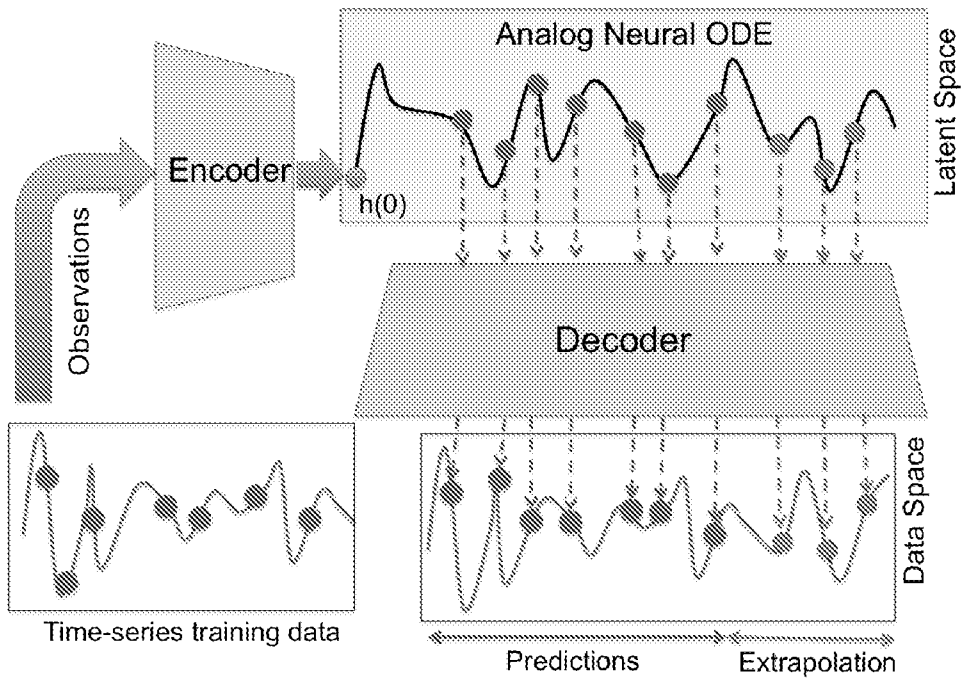

Figure 50

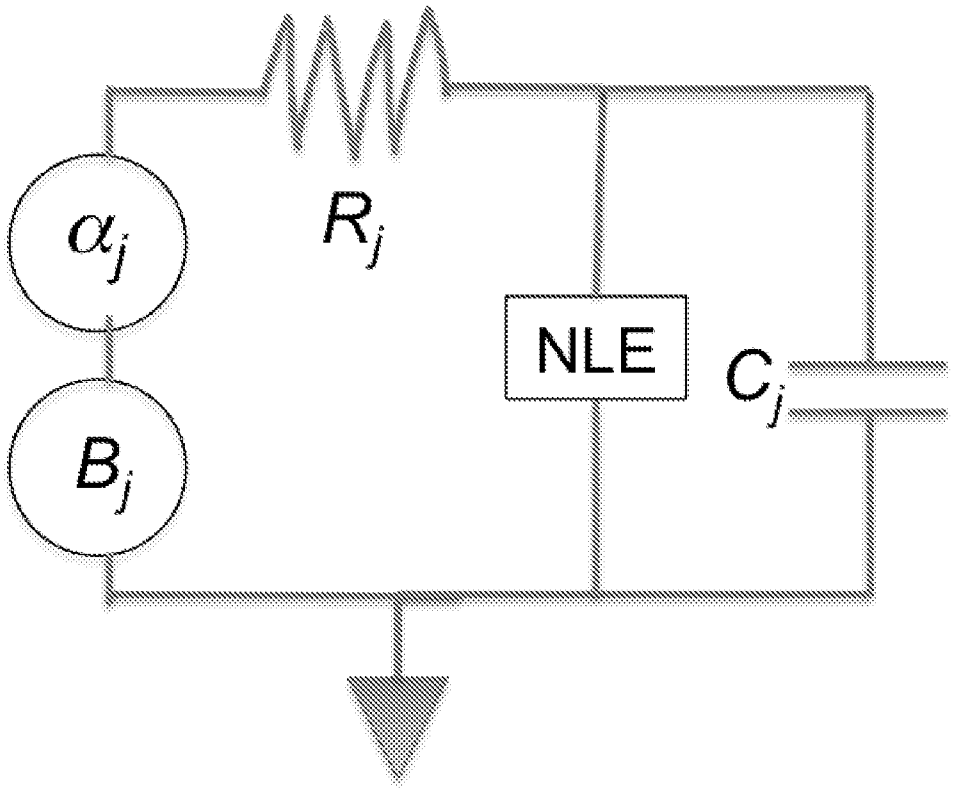

Figure 51

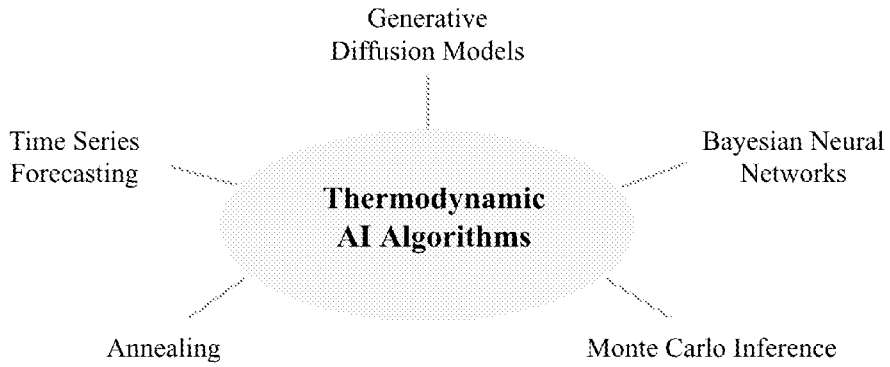

Figure 52

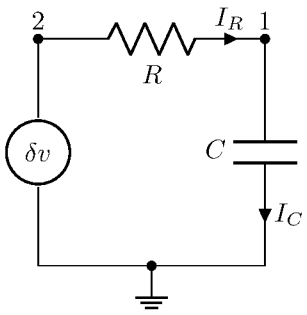

Figure 53

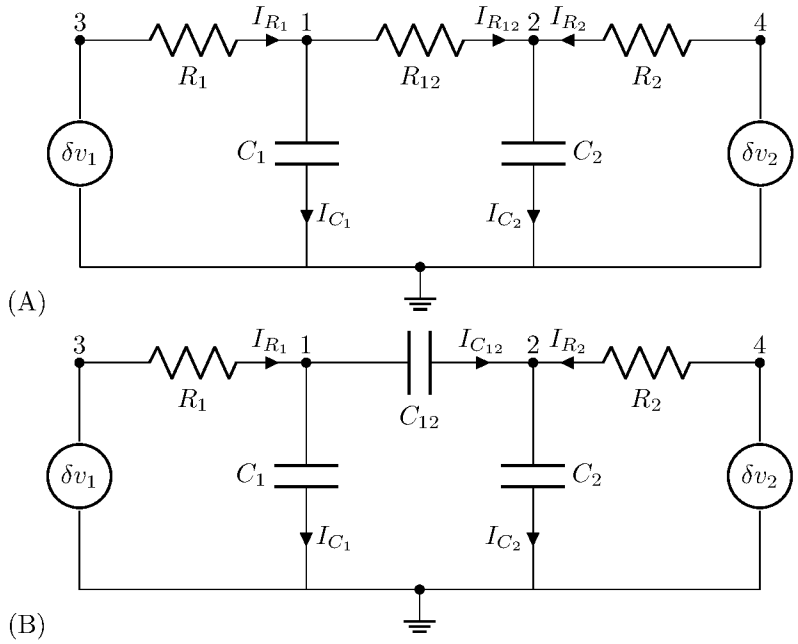

Figure 54

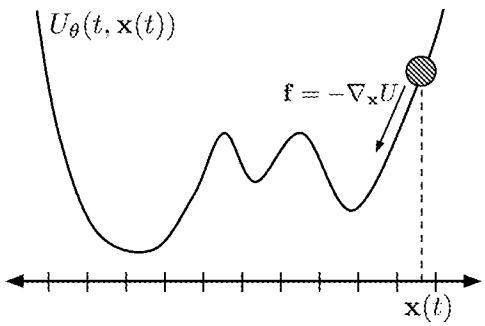

Figure 55

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 23/81816

A. CLASSIFICATION OF SUBJECT MATTER

IPC - INV. G05B 13/02, G05B 13/04 (2024.01)
ADD. G06N 7/04, G06N 7/06, G06N 20/00 (2024.01)

CPC - INV. G05B 13/0265, G05B 13/04, G05B 17/02, G06N 7/043
ADD. G05B 13/0205, G06N 20/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
See Search History document

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
See Search History document

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
See Search History document

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y --- A	US 2021/0397955 A1 (ROBERT BOSCH GMBH) 23 December 2021 (23.12.2021), para [0026], [0027], [0062], [0114]-[0119], [0128]	1-3, 5, 6, 11 --- 4, 7-10, 12-31
Y --- A	SOHL-DICKSTEIN et al., "Deep unsupervised learning using nonequilibrium thermodynamics." International conference on machine learning. PMLR, 2015, Fig 1; pg 1-5 [online] <https://proceedings.mlr.press/v37/sohl-dickstein15.pdf>	1-3, 5, 6, 11 --- 4, 7-10, 12-31
Y --- A	WO 2018/235004 A1 (SENDYNE CORPORATION) 27 December 2018 (27.12.2018), pg 1-6	11 --- 7-9, 12-30
A	US 2019/0113438 A1 (THE GENERAL HOSPITAL CORPORATION) 18 April 2019 (18.04.2019), entire document	1-31
A	US 2017/0169276 A1 (THE BOARD OF REGENTS OF THE UNIVERSITY OF TEXAS SYSTEM) 15 June 2017 (15.06.2017), entire document	1-31
A	LOOK et al., "Differential bayesian neural nets." arXiv preprint arXiv:1912.00796 (2019), entire document [online] <https://arxiv.org/pdf/1912.00796.pdf>	1-31
A	WO 2021/035038 A1 (THE GENERAL HOSPITAL CORPORATION) 25 February 2021 (25.02.2021), entire document	1-31

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"D" document cited by the applicant in the international application
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
27 March 2024	APR 18 2024
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-8300	Authorized officer Kari Rodriquez Telephone No. PCT Helpdesk: 571-272-4300

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 23/81816

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:
This application contains the following inventions or groups of inventions which are not so linked as to form a single general inventive concept under PCT Rule 13.1. In order for all inventions to be searched, the appropriate additional search fees must be paid.

Group I: Claims 1-31 drawn to a system for executing a generative diffusion model on a dataset.

Group II: Claims 32-56, drawn to a system for executing a Bayesian neural network.

Group III: Claims 57-64, drawn to a system for executing a neural ordinary differential equation (ODE).

Group IV: Claims 65-75, drawn to processor configured to implement a Bayesian neural network.

--see extra sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:
1-31

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 23/81816

Continuation of Box No. III – Observations where unity of invention is lacking

The inventions listed in the above-mentioned groups do not relate to a single general inventive concept under PCT Rule 13.1 because, under PCT Rule 13.2, they lack the same or corresponding special technical features for the following reasons:

Special Technical Features
Group I includes the special technical feature of a corresponding feature of the dataset, a tunable diffusion term and a tunable drift term, not included in the other groups.

Group II includes the special technical feature of a corresponding weight in the Bayesian neural network, not included in the other groups.

Group III includes the special technical feature of a hidden layer unit of the neural ODE, not included in the other groups.

Group IV includes the special technical feature of an analog weight diffuser having stochastic dynamics that generate a trajectory of weight values, not included in the other groups.

Common Technical Features:
The only technical features shared by Groups I-III that would otherwise unify the groups, are a physical system having a plurality of degrees of freedom and a continuous state variable that evolves according to a corresponding differential equation having a diffusion term and a drift term. However, these shared technical features do not represent a contribution over prior art, because the shared technical features are disclosed by US 2019/0113438 A1 to THE GENERAL HOSPITAL CORPORATION (hereinafter General).

The only additional technical feature shared by Groups II and IV that would otherwise unify the groups, is a Bayesian neural network. However, this shared technical feature does not represent a contribution over prior art, because the shared technical feature is disclosed by US 2017/0169276 A1 to The Board of Regents of The University of Texas System (hereinafter Regents).

General discloses a physical system having a plurality of degrees of freedom (para [0093]: a multivariate distribution of single-cell optical scatter properties that relate to single-cell size, cytoplasmic granularity, nuclear morphology, and other characteristics) and a continuous state variable (probability distribution) that evolves according to a corresponding differential equation having a diffusion term and a drift term (para [0041], [0093]-[0095]: Probability distribution for the lymphocyte population at the healthy and diseased state, contour plot shows how the model captures the trajectory of the probability distribution for a WBC subpopulation based on drift and diffusion...partial differential equation that describes this sort of temporal evolution of a probability density function under forces of drift and diffusion). Regents discloses a Bayesian neural network (para [0253]). It would have been obvious to one of ordinary skill in the art to use a Bayesian neural network of Regents with the physical system of General improve the classification performance of the system.

As the common technical features were known in the art, these cannot be considered special technical features that would otherwise unify the groups.

Therefore, Groups I-IV lack unity under PCT Rule 13.