Correspondence between neuroevolution and gradient descent

Stephen Whitelam[1],* Viktor Selin[2], Sang-Won Park[1], and Isaac Tamblyn[2,3,4]†
[1] *Molecular Foundry, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA*
[2] *Department of Physics, University of Ottawa, ON, K1N 6N5, Canada*
[2] *National Research Council of Canada, Ottawa, ON K1N 5A2, Canada*
[3] *Vector Institute for Artificial Intelligence, Toronto, ON M5G 1M1, Canada*

We show analytically that training a neural network by stochastic mutation or "neuroevolution" of its weights is equivalent, in the limit of small mutations, to gradient descent on the loss function in the presence of Gaussian white noise. Averaged over independent realizations of the learning process, neuroevolution is equivalent to gradient descent on the loss function. We use numerical simulation to show that this correspondence can be observed for finite mutations. Our results provide a connection between two distinct types of neural-network training, and provide justification for the empirical success of neuroevolution.

I. INTRODUCTION

In broad terms there are two types of method used to train neural networks, divided according to whether or not they explicitly evaluate gradients of the loss function. Gradient-based methods descend from the backpropagation algorithm [1–5]. In simple gradient descent, the parameters (weights and biases) $\boldsymbol{x}$ of a network evolve with time according to the prescription $\dot{\boldsymbol{x}} = -\alpha\nabla U(\boldsymbol{x})$, where α is a learning rate and $\nabla U(\boldsymbol{x})$ is the gradient of a loss function $U(\boldsymbol{x})$ with respect to the network parameters. The non-gradient-based methods are stochastic processes in which changes to a neural network are proposed and accepted with certain probabilities, and are related to Monte Carlo [6, 7] and genetic algorithms [8–10]. Here we focus on perhaps the simplest way of stochastically training a neural network, mutation or "neuroevolution" of its parameters [11] [10, 12–15]. Both gradient-based and non-gradient-based methods have been used to train neural networks for a variety of applications, and, where comparison exists, perform similarly well [14, 16–18].

Here we point out that there is a fundamental connection between gradient descent and neuroevolution. The connection follows from one identified in the 1990s between the overdamped Langevin dynamics and Metropolis Monte Carlo dynamics of a particle in an external potential [19, 20]. In the limit of small Monte Carlo trial moves, those things are equivalent. Similarly, neuroevolution in the limit of small mutations is equivalent to noisy gradient descent on the loss function. Specifically, let us propose a mutation $\boldsymbol{x} \to \boldsymbol{x}+\boldsymbol{\epsilon}$ of all neural-network parameters, where $\boldsymbol{\epsilon}$ is a set of independent Gaussian random numbers of zero mean and variance σ^2, and let us accept the proposal with the Metropolis probability $\min(1, \exp[-\beta\Delta U])$, a common choice in the physics literature [6, 7, 21]. Here β is a reciprocal evolutionary temperature, and ΔU is the change of the loss function under the proposal. When $\beta \neq \infty$, the weights of the network evolve, to leading order in σ, as $\dot{\boldsymbol{x}} = -(\beta\sigma^2/2)\nabla U(\boldsymbol{x})$ plus Gaussian white noise. Averaged over independent realizations of the learning process, simple neuroevolution is therefore equivalent to simple gradient descent, with learning rate $\beta\sigma^2/2$. In the limit $\beta = \infty$, where mutations are only accepted if the loss function does not increase, weights under neuroevolution evolve instead as $\dot{\boldsymbol{x}} = -(\sigma/\sqrt{2\pi})|\nabla U(\boldsymbol{x})|^{-1}\nabla U(\boldsymbol{x})$ plus Gaussian white noise, which corresponds to a form of gradient descent on $U(\boldsymbol{x})$.

We summarize this neuroevolution-gradient descent correspondence in Section II, and derive it in Section III. Our derivation follows the ideas developed in Ref. [20], with changes made to account for the different set-up [22] and proposal rates considered in that work and ours, and our interest in the limit $\beta = \infty$. It is natural to associate the state $\boldsymbol{x}$ of the neural network with the position of a particle in a high-dimensional space, and the loss function $U(\boldsymbol{x})$ with an external potential. The result is a rewriting of the correspondence between Langevin dynamics and Monte Carlo dynamics as a correspondence between the simplest forms of gradient descent and neuroevolution. Just as the Langevin-Monte Carlo correspondence provides a basis for understanding why Monte Carlo simulations of particles can approximate real dynamics [23–29], so the neuroevolution-gradient descent correspondence shows how we can effectively perform gradient descent on the loss function without explicit calculation of gradients. The correspondence holds exactly only in the limit of vanishing mutation scale, but we use numerics to show in Section IV that it can be observed for neuroevolution done with finite mutations and gradient descent enacted with a finite timestep. We conclude in Section V.

II. SUMMARY OF MAIN RESULTS

In this section we summarize the main analytic results of this paper.

Consider a neural network with N parameters (weights and biases) $\boldsymbol{x} = \{x_1, \ldots, x_N\}$, and a loss $U(\boldsymbol{x})$ that is a function of the network parameters. The precise archi-

* swhitelam@lbl.gov
† isaac.tamblyn@nrc.ca

tecture of the network is not important. The loss function may also depend upon other parameters, such as a set of training data, as in supervised learning, or a set of decisions and states, as in reinforcement learning; the correspondence we shall describe applies regardless.

A. Gradient descent

Under the simplest form of gradient descent, the parameters x_i of the network evolve according to numerical integration of

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = -\alpha \frac{\partial U(\boldsymbol{x})}{\partial x_i}. \tag{1}$$

Here time t measures the progress of training, and α is the learning rate [1–5].

B. Neuroevolution

Now consider training the network by neuroevolution, defined by the following Monte Carlo protocol.

1. Initialize the neural-network parameters $\boldsymbol{x}$ and calculate the loss function $U(\boldsymbol{x})$. Set time $t = 0$.

2. Propose a change (or "mutation") of each neural-network parameter by an independent Gaussian random number of zero mean and variance σ^2, so that

$$\boldsymbol{x} \to \boldsymbol{x} + \boldsymbol{\epsilon}, \tag{2}$$

   where $\boldsymbol{\epsilon} = \{\epsilon_1, \ldots, \epsilon_N\}$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

3. Accept the mutation with the Metropolis probability $\min\left(1, \mathrm{e}^{-\beta[U(\boldsymbol{x}+\boldsymbol{\epsilon})-U(\boldsymbol{x})]}\right)$, and otherwise reject it. In the latter case we return to the original neural network. The parameter β can be considered to be a reciprocal evolutionary temperature.

4. Increment time $t \to t + 1$, and return to step 2.

For non-infinite β, and in the limit of small mutation scale σ, the parameters of the neural network evolve under this procedure according to the Langevin equation

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = -\frac{\beta\sigma^2}{2}\frac{\partial U(\boldsymbol{x})}{\partial x_i} + \xi_i(t), \tag{3}$$

where ξ is a Gaussian white noise with zero mean and variance σ^2:

$$\langle \xi_i(t) \rangle = 0, \quad \langle \xi_i(t)\xi_j(t') \rangle = \sigma^2 \delta_{ij}\delta(t - t'). \tag{4}$$

Eq. (3) describes an evolution of the neural-network parameters x_i that is equivalent to gradient descent with learning rate $\alpha = \beta\sigma^2/2$ in the presence of Gaussian white noise. Averaging over independent stochastic trajectories of the learning process (starting from identical initial conditions) gives

$$\frac{\mathrm{d}\langle x_i \rangle}{\mathrm{d}t} = -\frac{\beta\sigma^2}{2}\frac{\partial U(\boldsymbol{x})}{\partial x_i}, \tag{5}$$

which has the same form as the gradient descent equation (1). Thus, when averaged over many independent realizations of the learning process, the neuroevolution procedure 1–4, with non-infinite β, is equivalent in the limit of small mutation scale to gradient descent on the loss function.

In the case $\beta = \infty$, where mutations are only accepted if the loss function does not increase, the parameters of the network evolve according to the Langevin equation

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = -\frac{\sigma}{\sqrt{2\pi}}\frac{1}{|\nabla U(\boldsymbol{x})|}\frac{\partial U(\boldsymbol{x})}{\partial x_i} + \eta_i(t), \tag{6}$$

where η is a Gaussian white noise with zero mean and variance $\sigma^2/2$:

$$\langle \eta_i(t) \rangle = 0, \quad \langle \eta_i(t)\eta_j(t') \rangle = \frac{\sigma^2}{2}\delta_{ij}\delta(t - t'). \tag{7}$$

The form (6) is different to (3), because the gradient in the first term is normalized by the factor $|\nabla U(\boldsymbol{x})| = \sqrt{\sum_{i=1}^{N} (\partial U(\boldsymbol{x})/\partial x_i)^2}$, which serves as an effective coordinate-dependent rescaling of the timestep, but (6) nonetheless describes a form of gradient descent on the loss function $U(\boldsymbol{x})$. Note that the drift term in (6) is of lower order in σ than the diffusion term (which is not the case for non-infinite β). In the limit of small σ, (6) describes an effective process in which uphill moves in loss cannot be made, consistent with the stochastic process from which it is derived.

Averaged over independent realizations of the learning process, (6) reads

$$\frac{\mathrm{d}\langle x_i \rangle}{\mathrm{d}t} = -\frac{\sigma}{\sqrt{2\pi}}\frac{1}{|\nabla U(\boldsymbol{x})|}\frac{\partial U(\boldsymbol{x})}{\partial x_i}. \tag{8}$$

The results (3) and (6) show that training a network by making random mutations of its parameters is, in the limit of small mutations, equivalent to noisy gradient descent on the loss function.

Writing $\dot{U}(\boldsymbol{x}) = \dot{\boldsymbol{x}} \cdot \nabla U(\boldsymbol{x})$, using (3) and (6), and averaging over noise shows the evolution of the mean loss function under neuroevolution to obey, in the limit of small σ,

$$\langle \dot{U}(\boldsymbol{x}) \rangle = \begin{cases} -\frac{\beta\sigma^2}{2}(\nabla U(\boldsymbol{x}))^2 & \text{if } \beta \neq \infty \\ -\frac{\sigma}{\sqrt{2\pi}}|\nabla U(\boldsymbol{x})| & \text{if } \beta = \infty \end{cases}, \tag{9}$$

equivalent to evolution under the noise-free forms of gradient descent (5) and (8).

In the following section we derive the correspondence described here; in Section IV we show that it can be observed numerically for non-vanishing mutations and finite integration steps.

III. DERIVATION OF (3) AND (6)

The master equation [30, 31] for the neuroevolution procedure defined in Section II B is

$$\partial_t P(\boldsymbol{x},t) = \int \mathrm{d}\boldsymbol{\epsilon}\,[P(\boldsymbol{x}-\boldsymbol{\epsilon},t)W_{\boldsymbol{\epsilon}}(\boldsymbol{x}-\boldsymbol{\epsilon}) - P(\boldsymbol{x},t)W_{\boldsymbol{\epsilon}}(\boldsymbol{x})]\,. \tag{10}$$

Here $P(\boldsymbol{x},t)$ is the probability that the network has the set of parameters $\boldsymbol{x}$ at time t (time being proportional to the number of steps of the procedure); $\int \mathrm{d}\boldsymbol{\epsilon} = \int_{-\infty}^{\infty}\mathrm{d}\epsilon_i\cdots\int_{-\infty}^{\infty}\mathrm{d}\epsilon_N$; and

$$W_{\boldsymbol{\epsilon}}(\boldsymbol{x}) = p(\boldsymbol{\epsilon})\min\left(1,\mathrm{e}^{-\beta[U(\boldsymbol{x}+\boldsymbol{\epsilon})-U(\boldsymbol{x})]}\right) \tag{11}$$

is the rate for going from the set of parameters $\boldsymbol{x}$ to the set of parameters $\boldsymbol{x}+\boldsymbol{\epsilon}$. The first factor in (11) is

$$p(\boldsymbol{\epsilon}) = \prod_{i=1}^{N} p(\epsilon_i) \quad \text{with} \quad p(\epsilon_i) = \frac{1}{\sqrt{2\pi\sigma^2}}\mathrm{e}^{-\frac{\epsilon_i^2}{2\sigma^2}}, \tag{12}$$

and accounts for the probability of proposing the set of Gaussian random numbers $\boldsymbol{\epsilon}$; the "min" function in (11) enforces the Metropolis acceptance rate.

We can pass from the master equation (10) to a Fokker-Planck equation by assuming a small mutation scale σ, and expanding the terms in (10) to second order in σ [30, 31]. For example,

$$P(\boldsymbol{x}-\boldsymbol{\epsilon},t) \approx \left(1-\boldsymbol{\epsilon}\cdot\nabla+\frac{1}{2}(\boldsymbol{\epsilon}\cdot\nabla)^2\right)P(\boldsymbol{x},t), \tag{13}$$

where $\boldsymbol{\epsilon}\cdot\nabla = \sum_{i=1}^{N}\epsilon_i\partial_i$ (note that $\partial_i \equiv \frac{\partial}{\partial x_i}$). Expanding $W_{\boldsymbol{\epsilon}}(\boldsymbol{x}-\boldsymbol{\epsilon})$ in the same way and collecting terms gives

$$\partial_t P(\boldsymbol{x},t) \approx -\int \mathrm{d}\boldsymbol{\epsilon}(\boldsymbol{\epsilon}\cdot\nabla)P(\boldsymbol{x},t)W_{\boldsymbol{\epsilon}}(\boldsymbol{x}) + \frac{1}{2}\int \mathrm{d}\boldsymbol{\epsilon}(\boldsymbol{\epsilon}\cdot\nabla)^2 P(\boldsymbol{x},t)W_{\boldsymbol{\epsilon}}(\boldsymbol{x}). \tag{14}$$

Taking the integrals inside the sums, (14) reads

$$\partial_t P(\boldsymbol{x},t) \approx -\sum_{i=1}^{N}\frac{\partial}{\partial x_i}\left(A_i(\boldsymbol{x})P(\boldsymbol{x},t)\right) + \frac{1}{2}\sum_{i,j=1}^{N}\frac{\partial^2}{\partial x_i\partial x_j}\left(B_{ij}(\boldsymbol{x})P(\boldsymbol{x},t)\right), \tag{15}$$

where

$$A_i(\boldsymbol{x}) \equiv \int \mathrm{d}\boldsymbol{\epsilon}\,\epsilon_i W_{\boldsymbol{\epsilon}}(\boldsymbol{x}), \tag{16}$$

and

$$B_{ij}(\boldsymbol{x}) \equiv \int \mathrm{d}\boldsymbol{\epsilon}\,\epsilon_i\epsilon_j W_{\boldsymbol{\epsilon}}(\boldsymbol{x}). \tag{17}$$

What remains is to calculate (16) and (17), which we do in different ways depending on the value of the evolutionary reciprocal temperature β.

A. Non-infinite β

In this section we consider non-infinite β, in which case we can evaluate (16) and (17) using the results of Refs. [19, 20] (making small changes in order to account for differences in proposal rates between those papers and ours).

Eq. (16) can be evaluated as

$$\begin{aligned}
A_i(\boldsymbol{x}) &= \int \mathrm{d}\boldsymbol{\epsilon}\,p(\boldsymbol{\epsilon})\epsilon_i\min\left(1,\mathrm{e}^{-\beta[U(\boldsymbol{x}+\boldsymbol{\epsilon})-U(\boldsymbol{x})]}\right) && (18)\\
&\approx \int \mathrm{d}\boldsymbol{\epsilon}\,p(\boldsymbol{\epsilon})\epsilon_i\min\left(1,1-\beta\boldsymbol{\epsilon}\cdot\nabla U\right) && (19)\\
&= \int \mathrm{d}\boldsymbol{\epsilon}\,p(\boldsymbol{\epsilon})\epsilon_i\Theta(-\boldsymbol{\epsilon}\cdot\nabla U)\\
&\quad + \int \mathrm{d}\boldsymbol{\epsilon}\,p(\boldsymbol{\epsilon})\epsilon_i(1-\beta\boldsymbol{\epsilon}\cdot\nabla U)\Theta(\boldsymbol{\epsilon}\cdot\nabla U) && (20)\\
&= \int \mathrm{d}\boldsymbol{\epsilon}\,p(\boldsymbol{\epsilon})\epsilon_i\\
&\quad - \beta\int \mathrm{d}\boldsymbol{\epsilon}\,p(\boldsymbol{\epsilon})\Theta(\boldsymbol{\epsilon}\cdot\nabla U)\sum_j \epsilon_i\epsilon_j\partial_j U && (21)\\
&= -\beta\int \mathrm{d}\boldsymbol{\epsilon}\,p(\boldsymbol{\epsilon})\Theta(\boldsymbol{\epsilon}\cdot\nabla U)\sum_j \epsilon_i\epsilon_j\partial_j U && (22)\\
&= -\frac{\beta}{2}\int \mathrm{d}\boldsymbol{\epsilon}\,p(\boldsymbol{\epsilon})\sum_j \epsilon_i\epsilon_j\partial_j U && (23)\\
&= -\frac{\beta\sigma^2}{2}\partial_i U. && (24)
\end{aligned}$$

In these expressions $\Theta(x) = 1$ if $x \geq 0$ and is zero otherwise. In going from (18) to (19) we have assumed that $\beta\boldsymbol{\epsilon}\cdot\nabla U(\boldsymbol{x})$ is small. This condition cannot be met for $\beta = \infty$; that case is treated in Section III B. In going from (20) to (21) we have used the result $\Theta(x)+\Theta(-x) = 1$; the first integral in (21) then vanishes by symmetry. The second integral can be evaluated as indicated in Ref. [20]. Upon a change of sign of the integration variables, $\boldsymbol{\epsilon} \to -\boldsymbol{\epsilon}$, the value of the integral is unchanged and it is brought to a form that is identical except for a change of sign of the argument of the theta function. Adding the two forms of the integral removes the theta functions, giving the form shown in (23), and dividing by 2 restores the value of the original integral. (23) can be evaluated using standard results of Gaussian integrals.

Eq. (17) can be evaluated in a similar way:

$$B_{ij}(\boldsymbol{x}) = \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon})\epsilon_i\epsilon_j \min\left(1, \mathrm{e}^{-\beta[U(\boldsymbol{x}+\boldsymbol{\epsilon})-U(\boldsymbol{x})]}\right) \quad (25)$$

$$\approx \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon})\epsilon_i\epsilon_j \min\left(1, 1-\beta\boldsymbol{\epsilon}\cdot\nabla U\right) \quad (26)$$

$$= \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon})\epsilon_i\epsilon_j \Theta(-\boldsymbol{\epsilon}\cdot\nabla U)$$

$$+ \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon})\epsilon_i\epsilon_j (1-\beta\boldsymbol{\epsilon}\cdot\nabla U)\Theta(\boldsymbol{\epsilon}\cdot\nabla U) \quad (27)$$

$$\approx \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon})\epsilon_i\epsilon_j \quad (28)$$

$$= \sigma^2\delta_{ij}. \quad (29)$$

The $\approx$ sign in (28) indicates that we have omitted terms of order σ^3.

Inserting (24) and (29) into (15) gives us, to second order in σ, the Fokker-Planck equation

$$\frac{\partial P(\boldsymbol{x},t)}{\partial t} \approx -\sum_{i=1}^{N}\frac{\partial}{\partial x_i}\left(-\frac{\beta\sigma^2}{2}\frac{\partial U(\boldsymbol{x})}{\partial x_i}P(\boldsymbol{x},t)\right) + \frac{1}{2}\sum_{i=1}^{N}\frac{\partial^2}{\partial x_i^2}\left(\sigma^2 P(\boldsymbol{x},t)\right). \quad (30)$$

This equation is equivalent [32] to the N Langevin equations [30, 31]

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = -\frac{\beta\sigma^2}{2}\frac{\partial U(\boldsymbol{x})}{\partial x_i} + \xi_i(t) \quad \forall i, \quad (31)$$

where ξ is a Gaussian white noise with zero mean and variance σ^2:

$$\langle\xi_i(t)\rangle = 0, \quad \langle\xi_i(t)\xi_j(t')\rangle = \sigma^2\delta_{ij}\delta(t-t'). \quad (32)$$

Eq. (31) describes an evolution of the neural-network parameters x_i that is equivalent to gradient descent with learning rate $\alpha = \beta\sigma^2/2$, plus a Gaussian white noise. Averaging over independent stochastic trajectories of the learning process (starting from identical initial conditions) gives

$$\frac{\mathrm{d}\langle x_i\rangle}{\mathrm{d}t} = -\frac{\beta\sigma^2}{2}\frac{\partial U(\boldsymbol{x})}{\partial x_i}, \quad (33)$$

which is equivalent to simple gradient descent on the loss function.

B. The case $\beta = \infty$

When $\beta = \infty$ we only accept mutations that do not increase the loss function. To treat this case we return to (16) and take the limit $\beta\to\infty$:

$$A_i(\boldsymbol{x}) = \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon})\epsilon_i\Theta(-\boldsymbol{\epsilon}\cdot\nabla U(\boldsymbol{x})). \quad (34)$$

We can make progress by introducing the integral form of the theta function (see e.g. [33]),

$$\Theta(-\boldsymbol{\epsilon}\cdot\nabla U(\boldsymbol{x})) = \int_{-\infty}^{0}\mathrm{d}z\,\delta(z-\boldsymbol{\epsilon}\cdot\nabla U(\boldsymbol{x})) \quad (35)$$

$$= \int_{-\infty}^{0}\frac{\mathrm{d}z}{2\pi}\int_{-\infty}^{\infty}\mathrm{d}\omega\,\mathrm{e}^{\mathrm{i}\omega(z-\boldsymbol{\epsilon}\cdot\nabla U(\boldsymbol{x}))}.$$

Then (34) reads

$$A_i(\boldsymbol{x}) = \int_{-\infty}^{0}\frac{\mathrm{d}z}{2\pi}\int_{-\infty}^{\infty}\mathrm{d}\omega\,\mathrm{e}^{\mathrm{i}\omega z}\,G_i^{(1)}\prod_{j\neq i}G_j^{(0)}, \quad (36)$$

where the symbols

$$G_j^{(n)} \equiv \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty}\mathrm{d}\epsilon_j\,\epsilon_j^n\,\mathrm{e}^{-\epsilon_j^2/(2\sigma^2)-\mathrm{i}\omega\epsilon_j\partial_j U} \quad (37)$$

are standard Gaussian integrals. Upon evaluating them as

$$G_j^{(0)} = \mathrm{e}^{-\frac{1}{2}\sigma^2\omega^2(\partial_j U)^2} \quad (38)$$

and

$$G_i^{(1)} = -\mathrm{i}\omega\sigma^2(\partial_i U)\mathrm{e}^{-\frac{1}{2}\sigma^2\omega^2(\partial_i U)^2}, \quad (39)$$

(36) reads

$$A_i(\boldsymbol{x}) = -\int_{-\infty}^{0}\frac{\mathrm{d}z}{2\pi}\int_{-\infty}^{\infty}\mathrm{d}\omega\,\mathrm{i}\sigma^2\omega(\partial_i U)\mathrm{e}^{-\frac{1}{2}\omega^2\sigma^2|\nabla U|^2+\mathrm{i}\omega z}$$

$$= \int_{-\infty}^{0}\frac{\mathrm{d}z}{2\pi}\sqrt{2\pi}\frac{\partial_i U}{\sigma|\nabla U|^3}z\mathrm{e}^{-z^2/(2\sigma^2|\nabla U|^2)} \quad (40)$$

$$= -\frac{\sigma}{\sqrt{2\pi}}\frac{\partial_i U(\boldsymbol{x})}{|\nabla U(\boldsymbol{x})|}. \quad (41)$$

The form (41) is similar to (24) in that it involves the derivative of the loss function $U(\boldsymbol{x})$ with respect to x_i, but contains an additional normalization term, $|\nabla U|$.

In the limit $\beta\to\infty$, (17) reads

$$B_{ij}(\boldsymbol{x}) = \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon})\epsilon_i\epsilon_j\Theta(-\boldsymbol{\epsilon}\cdot\nabla U(\boldsymbol{x})) \quad (42)$$

$$= \frac{1}{2}\int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon})\epsilon_i\epsilon_j \quad (43)$$

$$= \frac{1}{2}\sigma^2\delta_{ij}, \quad (44)$$

upon applying the symmetry arguments used to evaluate (22). Eq. (44) is half the value of the corresponding term for the case $\beta\neq\infty$, Eq. (29), because one term corresponds to Brownian motion in unrestricted space, the other to Brownian motion on a half-space.

Inserting (41) and (44) into (15) gives a Fokker-Planck equation equivalent to the N Langevin equations

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = -\frac{\sigma}{\sqrt{2\pi}}\frac{1}{|\nabla U(\boldsymbol{x})|}\frac{\partial U(\boldsymbol{x})}{\partial x_i} + \eta_i(t) \quad \forall i, \quad (45)$$

where η is a Gaussian white noise with zero mean and variance $\sigma^2/2$:

$$\langle\eta_i(t)\rangle = 0, \quad \langle\eta_i(t)\eta_j(t')\rangle = \frac{1}{2}\sigma^2\delta_{ij}\delta(t-t'). \quad (46)$$

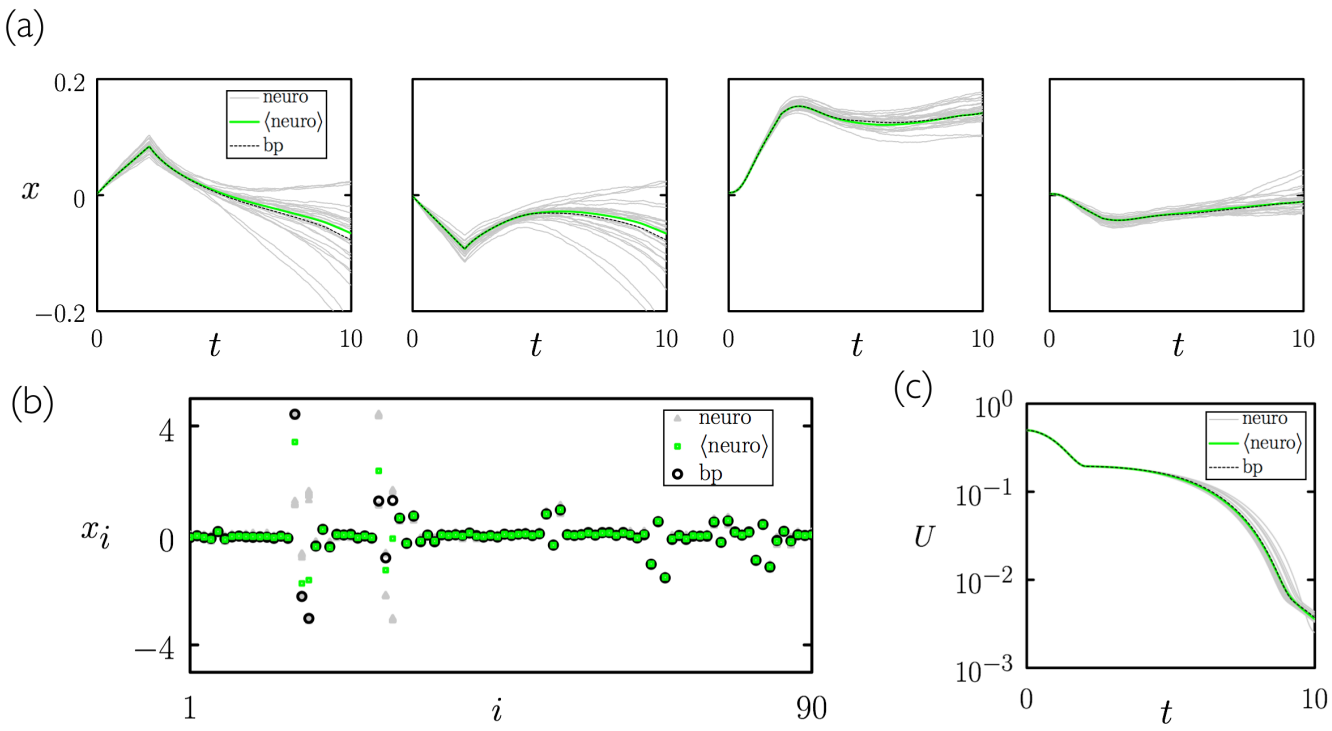

FIG. 1. Numerical illustration of the equivalence of gradient descent and neuroevolution in the case $\beta = \infty$. (a) Evolution with time of 4 of the 90 weights of the neural network (48) under modified gradient descent, Eq. (49) (black:"bp" stands for"backpropagation"), and under neuroevolution with mutation scale $\lambda = 1/10$ [see (51)]. Here and in subsequent panels we show 25 independent neuroevolution trajectories (grey) and the average of 1000 independent trajectories (green). (b) All weights at time $t = 10$ under the two methods. (c) Loss as a function of time.

IV. NUMERICAL ILLUSTRATION OF THE NEUROEVOLUTION-GRADIENT DESCENT CORRESPONDENCE

In this section we illustrate the neuroevolution-gradient descent correspondence for the case $\beta = \infty$. In this case the only requirement for correspondence is that the mutation scale σ is small enough that correction terms neglected in the expansion leading to (15) and (45) are small. The required range of σ is difficult to know in advance, but straightforward to determine empirically.

We consider a simple supervised-learning problem in which we train a neural network to express the function $f_0(\theta) = \sin(2\pi\theta)$ on the interval $\theta \in [0, 1)$. We calculated the loss using $K = 1000$ points on the interval,

$$U(\boldsymbol{x}) = \frac{1}{K}\sum_{j=0}^{K-1}\left[f_{\boldsymbol{x}}(j/K) - f_0(j/K)\right]^2, \qquad (47)$$

where

$$f_{\boldsymbol{x}}(\theta) = \sum_{i=0}^{M-1} x_{3i+1}\tanh(x_{3i+2}\theta + x_{3i+3}) \qquad (48)$$

is the output of a single-layer neural network with one input node, one output node, $M = 30$ hidden nodes, and $N = 3M$ parameters x_i. These parameters are initially chosen to be Gaussian random numbers with zero mean and variance $\sigma_0^2 = 10^{-4}$; we used the same randomly-initialized network for all simulations (conveniently done using the same seed for the pseudo-random number generator used to generate the initial network).

We performed modified gradient descent with learning rate $\alpha = 10^{-5}$, using Euler integration of the noise-free version of Eq. (6). Explicitly, we updated all weights x_i at each timestep $t_{\rm bp} = 1, 2, \ldots$ according to the prescription

$$x_i(t_{\rm bp}+1) = x_i(t_{\rm bp}) - \frac{\alpha}{|\nabla U(\boldsymbol{x})|}\frac{\partial U(\boldsymbol{x})}{\partial x_i}, \qquad (49)$$

where

$$\frac{\partial U(\boldsymbol{x})}{\partial x_i} = \frac{2}{K}\sum_{j=0}^{K-1}\left[f_{\boldsymbol{x}}(j/K) - f_0(j/K)\right]\frac{\partial f_{\boldsymbol{x}}(j/K)}{\partial x_i}, \qquad (50)$$

and

$$\frac{\partial f_{\boldsymbol{x}}(\theta)}{\partial x_i} = \begin{cases} \tanh(\theta x_{i+1} + x_{i+2}) & \text{if } i \bmod 3 = 1 \\ \theta x_{i-1}\,\mathrm{sech}^2(\theta x_i + x_{i+1}) & \text{if } i \bmod 3 = 2 \\ x_{i-2}\,\mathrm{sech}^2(\theta x_{i-1} + x_i) & \text{if } i \bmod 3 = 0 \end{cases}.$$

Recall that $|\nabla U(\boldsymbol{x})| = \sqrt{\sum_{i=1}^{N}(\partial U(\boldsymbol{x})/\partial x_i)^2}$.

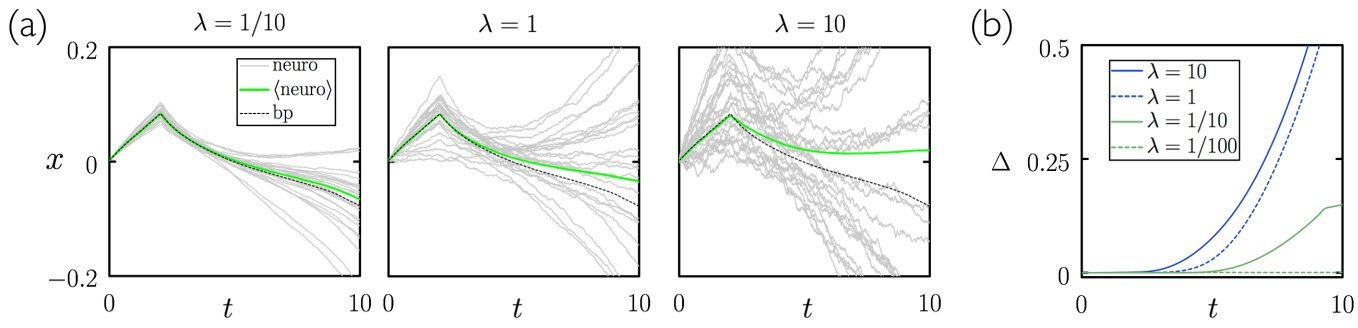

FIG. 2. The smaller the neuroevolution mutation scale, the closer the neuroevolution-gradient descent correspondence. (a) Time evolution of a single neural-network weight, for three different neuroevolution mutation scales [see (51)]. (b) The quantity $\Delta(t)$, Eq. (52), a measure of the similarity of networks evolved under gradient descent and neuroevolution.

We did neuroevolution following the Monte Carlo procedure described in Section II B, in the limit $\beta = \infty$, i.e. we accepted only moves that did not increase the loss function. We chose the mutation scale

$$\sigma = \lambda\alpha\sqrt{2\pi}, \tag{51}$$

where λ is a parameter. According to (6) and (49), this prescription sets the neuroevolution timescale $t_{\rm neuro}$ to be a factor λ times that of the modified gradient descent timescale. Thus one neuroevolution step correponds to λ integration steps of the gradient descent procedure. In figures, we compare modified gradient descent with neuroevolution as a function of common (scaled) time $t = \alpha t_{\rm bp} = \alpha\lambda t_{\rm neuro}$.

In Fig. 1 (a) we show the evolution of individual weights under neuroevolution and modified gradient descent (weights are distinguishable because they always have the same initial values). The correspondence predicted analytically can be seen numerically: individual neuroevolution trajectories (gray) fluctuate around the gradient descent result (black); and when averaged over individual trajectories, the results of neuroevolution (green) approximate those of gradient descent. In Fig. 1(b) we show the individual and averaged values of the weights of neuro-evolved networks at time $t = 10$ compared to those of modified gradient descent. In general, the weights generated by averaging over neuroevolution trajectories approximate those of gradient descent, with some discrepancy seen in the values of the largest weights. In Fig. 1(c) we show loss functions under neuroevolution and gradient descent. As predicted by (9), averaged neuroevolution and gradient descent are equivalent.

In Fig. 2(a) we show the time evolution of a single weight of the network under modified gradient descent and neuroevolution, the latter for three sizes of mutation step σ. As σ increases, the size of fluctuations of individual trajectories about the mean increase, as predicted by (6). As a result, more trajectories are required to estimate the average, and for fixed number of trajectories (as used here) the estimated average becomes less precise. In addition, as σ increases the assumptions underlying the correspondence derivation eventually break down, in which case the neuroevolution average will not converge to the gradient descent result even as more trajectories are sampled.

In Fig. 2(c) we show the quantity

$$\Delta(t) \equiv \frac{1}{N}\sum_{i=1}^{N}\left(x_i^{\rm bp}(t) - \langle x_i^{\rm neuro}(t)\rangle\right)^2, \tag{52}$$

which is a measure of the extent to which networks evolved under gradient descent and (averaged) neuroevolution differ. Here $x_i^{\rm bp}(t)$ is the time evolution of neural-network parameter i under modified gradient descent, Eq. (49), and $\langle x_i^{\rm neuro}(t)\rangle$ is the mean value of neural-network parameter i over the ensemble of neuroevolution trajectories. The smaller is the neuroevolution step size, the smaller is $\Delta(t)$, and the closer the neuroevolution-gradient descent correspondence.

In Fig. 3 we show the evolution with time of the loss function for different mutation scales, the trend being similar to that of the weights shown in Fig. 2.

V. CONCLUSIONS

We have shown analytically that training a neural network by neuroevolution of its weights is equivalent, in the limit of small mutation scale, to noisy gradient descent on the loss function. Conditioning neuroevolution on the Metropolis acceptance criterion at finite evolutionary temperature is equivalent to a noisy version of simple gradient descent, while at infinite reciprocal evolutionary temperature the procedure is equivalent to a different form of noisy gradient descent on the loss function. Averaged over noise, the evolutionary procedures therefore correspond to gradient descent on the loss function. This correspondence is described by Equations (3), (5), (6) and (8).

Our results provide a connection between two apparently dissimilar types of neural-network training. They also provide a foundation for understanding the

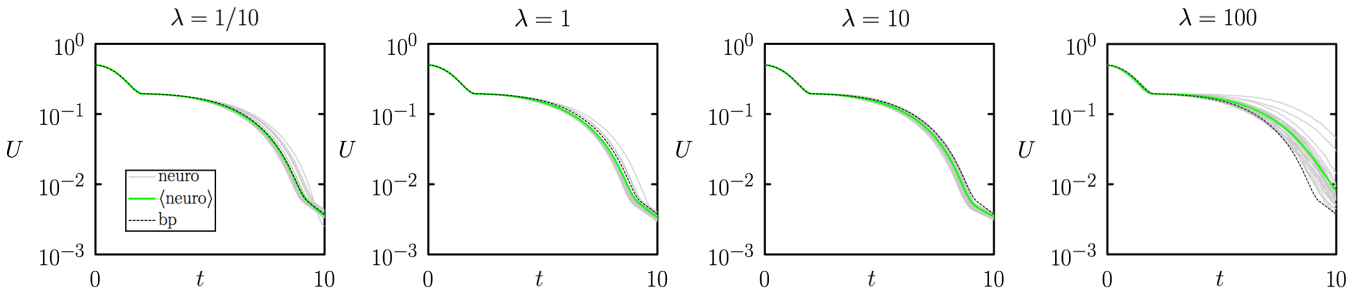

FIG. 3. Similar to Fig. 2(a), but showing the evolution of the loss function $U(\boldsymbol{x})$ with time. For small mutations [see (51)] the neuroevolution-gradient descent correspondence is apparent; for larger mutations the dynamics of neuroevolution and gradient descent are different, but neurevolution can still learn. For sufficiently large mutations neuroevolution will cease to learn.

empirically-observed ability of stochastic algorithms to compete with gradient-based methods [14, 16–18]. The neuroevolution-gradient descent equivalence results from an unweighted average over (i.e. the typical behavior of) a population of isolated individuals on which the mutation process acts independently. The mean loss function under neuroevolution is equal to that generated by noise-free gradient descent [see Eq. (9)]. By definition, there exist individuals in a neuroevolution population whose loss functions are smaller than the mean, and are therefore smaller than the loss function corresponding to noise-free gradient descent (see e.g. Fig. 1(c)). This argument is also relevant to genetic algorithms that periodically clone and eliminate members of a neuroevolution population. Consider a population of many realizations of the neuroevolution process, each evolved for a single mutation step. For small mutations, the portion of this population for which the loss function does not increase upon making the step is governed by Eq. (6), and so contains individuals whose loss functions lie below that produced by noise-free gradient descent. Choosing those individuals as the starting point for subsequent mutations will naturally probe the small-loss tail of the neuroevolution process.

The correspondence we have described is formally exact only in the limit of zero mutation scale, but we have shown that it can be observed numerically for nonzero mutations. Indeed, several dynamical regimes are contained within the neurevolution master equation (10), according to the scale σ of mutations [34]: for vanishing σ, neurevolution is formally noisy gradient descent on the loss function; for small but nonvanishing σ it approximates noisy gradient descent enacted by explicit integration with a finite timestep; for larger σ it enacts a distinct dynamics, but one that can still learn; and for sufficiently large σ the steps taken are too large for learning to occur on accessible timescales. An indication of these various regimes can be seen in Figs. 2 and 3.

Separate from the question of its precise temporal evolution, the master equation (10) has a well-defined stationary distribution $\rho_0(\boldsymbol{x})$. Requiring the brackets on the right-hand of (10) to vanish ensures that $P(\boldsymbol{x}, t) \to \rho_0(\boldsymbol{x})$ becomes independent of time. Inserting (11) into (10) and requiring normalization of $\rho_0(\boldsymbol{x})$ reveals the stationary distribution to be the Boltzmann one, $\rho_0(\boldsymbol{x}) = \mathrm{e}^{-\beta U(\boldsymbol{x})} / \int \mathrm{d}\boldsymbol{x}' \mathrm{e}^{-\beta U(\boldsymbol{x}')}$. For non-infinite β the neuroevolution procedure is ergodic, and this distribution will be sampled given sufficiently long simulation time. For $\beta \to \infty$ we have $\rho_0(\boldsymbol{x}) \to \delta(U(\boldsymbol{x}) - U_0)$, where U_0 is the global energy minimum; in this case the system is not ergodic (moves uphill in $U(\boldsymbol{x})$ are not allowed) and there is no guarantee of reaching this minimum.

The neuroevolution-gradient descent corresponence we have identified follows from that between the overdamped Langevin dynamics and Metropolis Monte Carlo dynamics of a particle in an external potential [19, 20]. Our work therefore adds to the existing set of connections between machine learning and statistical mechanics [35, 36], and continues a trend in machine learning of making use of old results: the stochastic and deterministic algorithms considered here come from the 1950s [6, 7] and 1980s [1–4], and are connected by ideas developed in the 1990s [19, 20].

Acknowledgments – This work was performed as part of a user project at the Molecular Foundry, Lawrence Berkeley National Laboratory, supported by the Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC02–05CH11231. This work used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. I.T. acknowledges funding from the National Science and Engineering Council of Canada and carried out work at the National Research Council of Canada under the auspices of the AI4D Program.

[1] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin, "Backpropagation: The basic theory," Backpropagation: Theory, architectures and appli-

cations , 1–34 (1995).
[2] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, "Learning representations by back-propagating errors," Nature **323**, 533–536 (1986).
[3] Robert Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception* (Elsevier, 1992) pp. 65–93.
[4] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, "Backpropagation applied to handwritten zip code recognition," Neural computation **1**, 541–551 (1989).
[5] Yves Chauvin and David E Rumelhart, *Backpropagation: theory, architectures, and applications* (Psychology press, 1995).
[6] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller, "Equation of state calculations by fast computing machines," The Journal of Chemical Physics **21**, 1087–1092 (1953).
[7] W Keith Hastings, "Monte Carlo sampling methods using markov chains and their applications," Biometrika **5 7**, 97–109 (1970).
[8] John H Holland, "Genetic algorithms," Scientific american **267**, 66–73 (1992).
[9] David B Fogel and Lauren C Stayton, "On the effectiveness of crossover in simulated evolutionary optimization," BioSystems **32**, 171–182 (1994).
[10] David J Montana and Lawrence Davis, "Training feedforward neural networks using genetic algorithms." in *IJCAI*, Vol. 89 (1989) pp. 762–767.
[11] The term neuroevolution is also used to describe mutations of network topology [12]; here we use the term to describe mutations of the weights and biases of a network.
[12] Dario Floreano, Peter Dürr, and Claudio Mattiussi, "Neuroevolution: from architectures to learning," Evolutionary intelligence **1**, 47–62 (2008).
[13] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune, "Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," arXiv preprint arXiv:1712.06567 (2017).
[14] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," arXiv preprint arXiv:1703.03864 (2017).
[15] Stephen Whitelam and Isaac Tamblyn, "Learning to grow: Control of material self-assembly using evolutionary reinforcement learning," Physical Review E **101**, 052604 (2020).
[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, "Playing Atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602 (2013).
[17] Gregory Morse and Kenneth O Stanley, "Simple evolutionary optimization can rival stochastic gradient descent in neural networks," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (2016) pp. 477–484.
[18] Xingwen Zhang, Jeff Clune, and Kenneth O Stanley, "On the relationship between the OpenAI evolution strategy and stochastic gradient descent," arXiv preprint arXiv:1712.06564 (2017).
[19] K Kikuchi, M Yoshida, T Maekawa, and H Watanabe, "Metropolis Monte Carlo method as a numerical technique to solve the fokker-planck equation," Chemical Physics Letters **185**, 335–338 (1991).
[20] K Kikuchi, M Yoshida, T Maekawa, and H Watanabe, "Metropolis Monte Carlo method for brownian dynamics simulation generalized to include hydrodynamic interactions," Chemical Physics letters **196**, 57–61 (1992).
[21] Daan Frenkel and Berend Smit, *Understanding molecular simulation: from algorithms to applications*, Vol. 1 (Academic Press, 2001).
[22] In effect we work with a single particle in a high-dimensional space, rather than with many particles in three-dimensional space.
[23] Stephen Whitelam and Phillip L Geissler, "Avoiding unphysical kinetic traps in Monte Carlo simulations of strongly attractive particles," The Journal of Chemical Physics **127**, 154101 (2007).
[24] Alex W Wilber, Jonathan PK Doye, Ard A Louis, Eva G Noya, Mark A Miller, and Pauline Wong, "Reversible self-assembly of patchy particles into monodisperse icosahedral clusters," The Journal of Chemical Physics **127**, 08B618 (2007).
[25] Ludovic Berthier, "Revisiting the slow dynamics of a silica melt using Monte Carlo simulations," Physical Review E **76**, 011507 (2007).
[26] E Sanz and D Marenduzzo, "Dynamic Monte Carlo versus Brownian dynamics: A comparison for self-diffusion and crystallization in colloidal fluids," The Journal of Chemical Physics **132**, 194102 (2010).
[27] Stephen Whitelam, "Approximating the dynamical evolution of systems of strongly interacting overdamped particles," Molecular Simulation **37**, 606–612 (2011).
[28] Xiao Liu, John C Crocker, and Talid Sinno, "Coarse-grained Monte Carlo simulations of non-equilibrium systems," The Journal of Chemical Physics **138**, 244111 (2013).
[29] Lorenzo Rovigatti, John Russo, and Flavio Romano, "How to simulate patchy particles," The European Physical Journal E **41**, 59 (2018).
[30] Hannes Risken, "Fokker-planck equation," in *The Fokker-Planck Equation* (Springer, 1996) pp. 63–95.
[31] Nicolaas Godfried Van Kampen, *Stochastic processes in physics and chemistry*, Vol. 1 (Elsevier, 1992).
[32] The diffusion term is independent of $\boldsymbol{x}$, and so the choice of stochastic calculus is unimportant.
[33] Y Bar Sinai, *https://yohai.github.io/post/half-gaussian/* (2019).
[34] Choosing σ to be a parameter in a genetic search scheme would allow exploration of the dynamical diversity contained within Eq. (10).
[35] Andreas Engel and Christian Van den Broeck, *Statistical mechanics of learning* (Cambridge University Press, 2001).
[36] Yasaman Bahri, Jonathan Kadmon, Jeffrey Pennington, Sam S Schoenholz, Jascha Sohl-Dickstein, and Surya Ganguli, "Statistical mechanics of deep learning," Annual Review of Condensed Matter Physics (2020).