https://doi.org/10.1038/s44335-025-00049-x

Thermodynamic natural gradient descent

Kaelan Donatella[1,2], Samuel Duffield[1,2] ✉, Maxwell Aifer[1], Denis Melanson[1], Gavin Crooks[1] & Patrick J. Coles[1]

Second-order training methods have better convergence properties than gradient descent but are rarely used in practice for large-scale training due to their computational overhead. This can be viewed as a hardware limitation (imposed by digital computers). Here, we show that natural gradient descent (NGD), a second-order method, can have a similar computational complexity per iteration to a first-order method when employing appropriate hardware. We present a new hybrid digital-analog algorithm for training neural networks that is equivalent to NGD in a certain parameter regime but avoids prohibitively costly linear system solves. Our algorithm exploits the thermodynamic properties of an analog system at equilibrium, and hence requires an analog thermodynamic computer. The training occurs in a hybrid digital-analog loop, where the gradient and Fisher information matrix (or any other positive semi-definite curvature matrix) are calculated at given time intervals while the analog dynamics take place. We numerically demonstrate the superiority of this approach over state-of-the-art digital first- and second-order training methods on classification tasks and language model fine-tuning tasks.

With the rise of more sophisticated AI models, the cost of training them is exploding, as world-leading models now cost hundreds of millions of dollars to train. This issue is compounded by the ending of both Moore's Law and Dennard's Law for digital hardware[1], which impacts both the runtime and energy efficiency of such hardware. This highlights a need and an opportunity for specialized, unconventional hardware targeted at improving the efficiency of training AI models.

Moreover, conventional digital hardware can be viewed as limiting the range of training algorithms that a user may consider. Researchers are missing an opportunity to co-design novel optimizers to exploit novel hardware developments. Instead, relatively simplistic optimizers, such as stochastic gradient descent (SGD), Adam[2], and their variants[3], are among the most popular methods for training deep neural networks (DNNs) and other large AI models. More sophisticated optimizers are rarely used due to the associated computational overhead on digital hardware.

A clear example of this is second-order methods, which capture curvature information of the loss landscape. These methods, while theoretically more powerful in terms of convergence properties, remain computationally expensive and harder to use, blocking their adoption. For example, natural gradient descent (NGD)[4,5] involves calculating estimates of second-order quantities such as the Fisher information matrix and performing a costly linear system solve at every epoch. Some approximations to NGD, such as the Kronecker-factored approximate curvature (K-FAC)[6], have shown promise, and K-FAC has shown superior performance to Adam[7,8]. However, applying such methods to arbitrary neural network architectures remains difficult[9].

In this article, we present thermodynamic natural gradient descent (TNGD), a new method to perform second-order optimization. This method involves a hybrid digital-analog loop, where a GPU communicates with an analog thermodynamic computer. A nice feature of this paradigm is flexibility: the user provides their model architecture and the analog computer serves only to accelerate the training process. This is in contrast to many proposals to accelerate the inference workload of AI models with analog computing, where the model is hardwired into the hardware, and users are unable to change the model architecture as they seamlessly would by using their preferred software tools[10–13].

The analog computer in TNGD uses thermodynamic processes as a computational resource. Such thermodynamic devices have previously been proposed[14–18], have been theorized to exhibit runtime and energy efficiency gains[19,20], and have been successfully prototyped[21,22]. Our TNGD algorithm represents an instance of algorithmic co-design, where we propose a novel optimizer to take advantage of a novel hardware paradigm. TNGD exploits a physical Ornstein-Uhlenbeck process to implement the parameter update rule in NGD. It has a runtime per iteration scaling linearly in the number of parameters, and when properly parallelized it can be close to the runtime of first-order optimizers such as Adam and SGD. Hence, it is theoretically possible to achieve the computational efficiency of a first-order training method while still accounting for the curvature of the loss landscape with a second-order method. Moreover, our numerics show the competitiveness of TNGD with first-order methods for classification and extractive question-answering tasks.

There is a large body of theoretical research on natural gradient descent[4,5,23] arguing that NGD requires fewer iterations than SGD to converge to the same value of the loss in specific settings. While less is known about the theoretical convergence rate of Adam, there exists a large body of

[1]Normal Computing Corporation, New York, NY, USA. [2]These authors contributed equally: Kaelan Donatella, Samuel Duffield. ✉e-mail: sam@normalcomputing.ai

empirical evidence that NGD can converge in fewer iterations than Adam[6,8,24–27].

However, a single iteration of NGD is generally more computationally expensive than that of SGD or Adam, which have a per-iteration cost scaling linearly in the number of parameters N. NGD typically has a superlinear complexity in the number of parameters. K-FAC[6] aims to reduce this complexity and invokes a block-wise approximation of the curvature matrix, which may not always hold. While first introduced for multi-layer perceptrons, K-FAC has been applied to more complex architectures, such as recurrent neural networks[25] and transformers[8], where additional approximations have to be made and where the associated computational overhead can vary.

There has been significant effort and progress towards reducing the time- and space-complexity of operations used in the inference workload of AI models, e.g., a variety of "linear attention" blocks have been proposed[28–30]. However, there has been less focus on reducing the complexity of training methods. While various approaches are taken to accelerate training using novel hardware, these efforts typically aim to reduce the constant coefficients appearing in the time cost of computation. Especially relevant to our work, analog computing devices have been proposed to achieve reduced time and energy costs of training relative to available digital technology[10–13]. These devices are generally limited to training a neural network that has a specific architecture (corresponding to the structure of the analog device). To our knowledge, there has not yet been a proposal that leverages analog hardware to reduce the complexity of training algorithms such as NGD.

Given the existing results implying that fewer iterations are needed for NGD relative to other commonly used optimizers, we focus on reducing the per-iteration computational cost of NGD using a hybrid analog-digital algorithm to perform each parameter update. Our algorithm therefore, demonstrates that complexity can be improved in training (not only in inference), and moreover that the per-iteration complexity of NGD can be made similar to that of a first-order training method.

Results

Thermodynamic natural gradient descent

At a high level, TNGD combines the strength of GPUs (through auto-differentiation) with the strength of thermodynamic devices at solving linear systems. Regarding the latter,[19] showed that a thermodynamic device, called a stochastic processing unit (SPU), can solve a linear system $Ax = b$ with reduced computational complexity relative to standard digital hardware. The solution to the linear system is found by letting the SPU evolve under an Ornstein-Uhlenbeck (OU) process given by the following stochastic differential equation (SDE):

$$dx = -(Ax - b)dt + \mathcal{N}[0, 2\beta^{-1}\,dt], \tag{1}$$

where A is a positive matrix and β is a positive scalar (which can be seen as the inverse temperature of the noise). Operationally, one lets the SPU settle to its equilibrium state under the dynamics of (1), at which point x is distributed according to the Boltzmann distribution given by:

$$x \sim \mathcal{N}[A^{-1}b, \beta^{-1}A^{-1}]. \tag{2}$$

One can see that the first moment of this distribution is the solution to the linear system $Ax = b$. Exploiting this approach, TNGD involves a subroutine that estimates the solution to the linear system in the following definition of the natural gradient

$$\tilde{g} = F^{-1}\nabla\ell(\theta), \tag{3}$$

where F is the *Fisher information matrix*[5], and $\ell(\theta)$ is the objective function to be minimized over the parameters θ. See the Methods section for more details. For this particular linear system, the SDE in (1) becomes the following:

$$d\tilde{g}_{k,t} = -(F_{k-1}\tilde{g}_{k,t} - \nabla\ell_{k-1})dt + \mathcal{N}[0, 2\kappa_0 dt] \tag{4}$$

$$= -(J_{f,k-1}^{\top}H_{L,k-1}J_{f,k-1}\tilde{g}_{k,t} - \nabla\ell_{k-1})dt + \mathcal{N}[0, 2\kappa_0 dt] \tag{5}$$

with $\tilde{g}_{k,t}$ the value of the natural gradient estimate at time t and iteration k, κ_0 the variance of the noise. In the last step of (4), we used the generalized Gauss-Newton (GGN) approximation to the Fisher matrix which involves J_f the Jacobian of the forward function, and H_L the Hessian of the loss function (see Methods for definitions). Comparing (1) and (4), we see that in the equilibrium state (i.e. for large t), the mean of $\tilde{g}_{k,t}$ provides an estimate of the natural gradient, in other words:

$$\tilde{g}_k := \lim_{t\to\infty}\langle\tilde{g}_{k,t}\rangle = F_{k-1}^{-1}\nabla\ell_{k-1}. \tag{6}$$

The overall TNGD algorithm is illustrated in Fig. 1. Using the current parameter estimates θ_k, the GPU computes the matrices J_f and H_L, and the vector $\nabla\ell$, which can be accomplished efficiently using auto-differentiation. The matrices J_f, $J_f^{\top}$, and H_L, as well as the vector $\nabla\ell$, are uploaded to component values (see Supplementary Material) on the SPU, which is then allowed to equilibrate under the dynamics of (4). Next, samples are taken of $\tilde{g}_{k,t}$, and are sent from the SPU to the GPU, where samples are averaged to yield an estimate of $\tilde{g}_k$. Finally, the parameters are updated using the equation

$$\theta_{k+1} = \theta_k - \eta\tilde{g}_k, \tag{7}$$

and this process may be repeated until sufficient convergence is achieved (other update equations may also be employed, see Experiments).

Computational complexity and performance

The runtime complexity of TNGD and other second-order optimization (that do not make assumptions on the structure of the Fisher matrix, hence excluding K-FAC) algorithms is reported in Table 1. Specifically, thermodynamic NGD (TNGD) is compared to other NGD variants that solve the linear system in (3) using the conjugate gradient method[24] (NGD-CG) and using the Woodbury identity[26] (NGD-Woodbury), further details on these methods can be found in the Methods section. As explained, Thermodynamic NGD (TNGD) has a runtime and memory cost dominated by the construction and storage (before sending them off to the analog hardware) of the Jacobian of the forward function and the Hessian of the loss. The t factor denotes the analog runtime, and may be interpreted similarly to c for NGD-CG as a parameter controlling the approximation. For each optimizer the number of model calls is reported. For all optimizers except NGD-CG these calls can be easily parallelized thanks to vectorizing maps in PyTorch.

In Fig. 2 a comparison of the runtime per iteration of the four second-order optimizers considered is shown. Figure 2(a) shows the runtime as a function of the number of parameters N. The scaling of NGD as N^3 can be observed, and the NGD-CG data is close to flat, meaning the model calls parallelize well for the range of parameter count considered. The linear scaling of NGD-Woodbury and TNGD is also shown, although with a different overall behaviour due to parallelization and a much shorter runtime per iteration for TNGD. This shows that for the given range of N at $d_z = 20$, we can expect a 100 × speedup over second-order optimizers. Figure 2(b) shows the dependence of runtime on the output dimension d_z for the second-order optimizers. These results indicate that TNGD is most competitive for intermediate values of d_z. Finally, we note that with better hardware, the scaling with both N and d_z would be better, as the operations to construct the Hessian and Jacobian can be more efficiently parallelized for larger values. We note that Fig. 2(b) shows increasing scaling as d_z increases which is due to exhausting the parallelism capacity of the fixed GPU in

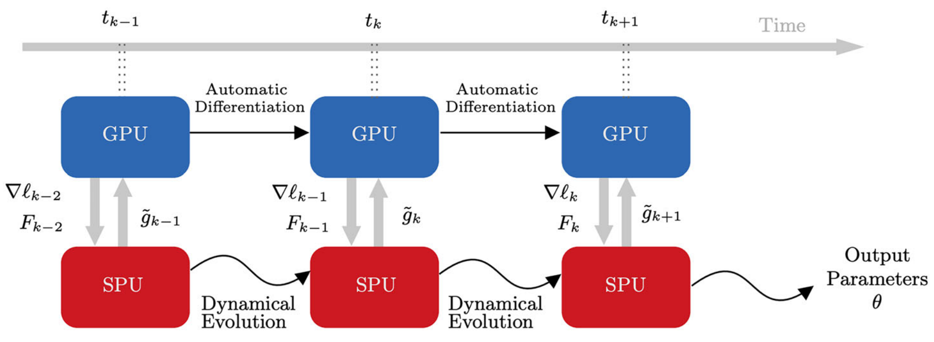

Fig. 1 | Overview of thermodynamic natural gradient descent (TNGD). A GPU that stores the model architecture and provides the gradient $\nabla\ \ell_k$ and Fisher matrix F_k (through its representation given by the Jacobian J_f and Hessian H_L matrices given by (17)) at step k is connected to a thermodynamic computer, called the stochastic processing unit (SPU). At times t_k, the estimate of the natural gradient $\widetilde{g}_k$ is sent to the GPU, which updates the parameters of the model and calculates gradients and curvature matrices for some new data batch (x_k, y_k). During digital auto-differentiation, the SPU undergoes dynamical evolution, either continuing to approach its steady-state or remaining in it. After some time, gradient $\nabla\ \ell_k$ and Fisher matrix F_k are sent to the SPU through a DAC and digital controllers. This modifies the dynamics of the SPU, and after some time interval, a new natural gradient estimate $\widetilde{g}_{k+1}$ is sent back to the GPU. Note that the time between two measurements $t_{k+1} - t_k$ need not be greater than the time between two auto-differentiation calls. The hybrid digital-thermodynamic process may be used asynchronously as shown in the diagram (where the time of measurement of $\widetilde{g}$ and upload of the gradient and Fisher matrix are not the same).

Table 1 | Runtime and memory complexity of optimizers considered in this paper

Optimizer	Runtime	Memory	Model calls
SGD/Adam	$O(bN)$	$O(N)$	1
NGD	$O(N^3 + bd_zN^2)$	$O(N^2)$	bd_z
NGD-CG	$O(cbN)$	$O(N)$	$2c$
NGD-Woodbury	$O(bd_z^2N + b^3d_z^3)$	$O(bd_zN + bd_z^2)$	bd_z
Thermodynamic NGD	$O(bd_zN + t)$	$O(bd_zN + bd_z^2)$	bd_z

All operations are per iteration. The first line corresponds to first-order optimizers that evaluate the gradient only, and apply diagonal rescalings and $O(N)$ operations to it only. Vanilla NGD (second line) includes the explicit storage and inversion of the GGN matrix as well as its construction, dominating the runtime and memory cost. NGD-CG (third line) can be performed by running c iterations, each dominated by GGN-vector products and has the same memory cost as first-order methods. NGD-Woodbury can be performed by constructing the matrix VU and using the formula given by (20). This results in a runtime cost dominated by constructing VU and inverting it, which also requires its storage.

calculating the Jacobian matrices (the dominating cost)—see Supplementary Material for further details.

In addition to the advantage in time- and energy-efficiency, TNGD has another advantage over NGD-CG in terms of stability. For some pathological linear systems, CG fails to converge and instead diverges. However, the thermodynamic algorithm is guaranteed to converge (on average) for any positive definite matrix. To see this, note that the mean of $\widetilde{g}_{k,t}$ evolves according to

$$\langle\widetilde{g}_{k,t}\rangle = \exp(-F_{k-1}t)(\widetilde{g}_{k,0} - F_{k-1}^{-1}\nabla\ell_{k-1}) + F_{k-1}^{-1}\nabla\ell_{k-1}. \tag{8}$$

There is still variance associated with the estimator of $\langle\widetilde{g}_{k,t}\rangle$, but the trajectory converges to the stationary distribution (2), with high probability in all cases. We also note that if we choose $\widetilde{g}_{k,0} = \nabla\ell_{k-1}$, we obtain a smooth interpolation between SGD ($t = 0$) and NGD ($t = \infty$).

Analog dynamics time t and delay time t_d

A key insight in the seminal work on iterative approaches to the linear system solve within NGD[24] is that the overall gradient descent algorithm can still perform well even if the iterative linear system solver (conjugate gradient) is only run for a small number of iterations. In terms of TNGD, this corresponds to a smaller dynamics time t and also motivates taking only a single sample that approaches a sample from (2) (which we apply in all experiments). As mentioned above, starting with $\widetilde{g}_{k,0} = \nabla\ell_{k-1}$ results in a starting point equivalent to SGD and as the trajectory evolves closer to (2), we expect the performance to improve but even the initial $t = 0$ represents a convergent algorithm. The effect of this hyperparameter t is investigated in Figs. 4 and 5 in the experiments section.

Furthermore, another hyperparameter arises from the delay time t_d, defined as the time between a measurement of θ_k and the update of the gradient and GGN on the device. As discussed in the Experiments section, a non-zero delay time is not necessarily detrimental to performance and can in fact, improve it. Setting $t_d = t$ would correspond a fully hybrid approach where both the analog device used to solve the linear system and the digital device used to evaluate the gradient have zero idle time, as depicted in Fig. 1.

Experiments

MNIST classification. We first consider the task of MNIST classification[31]. For our experiments, we use a simple convolutional neural network consisting of a convolutional layer followed by two feedforward layers, and we digitally simulate the TNGD algorithm (see Supplementary Material). The goal of these experiments is twofold: (1) to compare the estimated performance per runtime of TNGD against popular first-order optimizers such as Adam, and (2) to provide some insights on other features of TNGD, such as its performance as a function of the analog runtime t as well as its asynchronous execution as a function of the delay time t_d.

In Fig. 3(a), the training and test losses as a function of runtime for both Adam (measured) and TNGD (estimated) are presented. To estimate the TNGD runtime, we took into account results for its runtime per iteration as presented in the previous section, finding an overall 2 × runtime per iteration with respect to Adam for this problem on an A100 GPU. One can see from the figure that even while taking into account the difference in runtime per iteration, TNGD still outperforms Adam, especially at the initial stages of the optimization. Interestingly, it also generalizes better for the considered experimental setup. In Fig.3(b), the training and test accuracies are shown. We again see TNGD largely outperforming Adam, reaching the same training accuracy orders of magnitude faster, while also displaying a better test accuracy. These results are reminiscent of prior work on NGD[24],

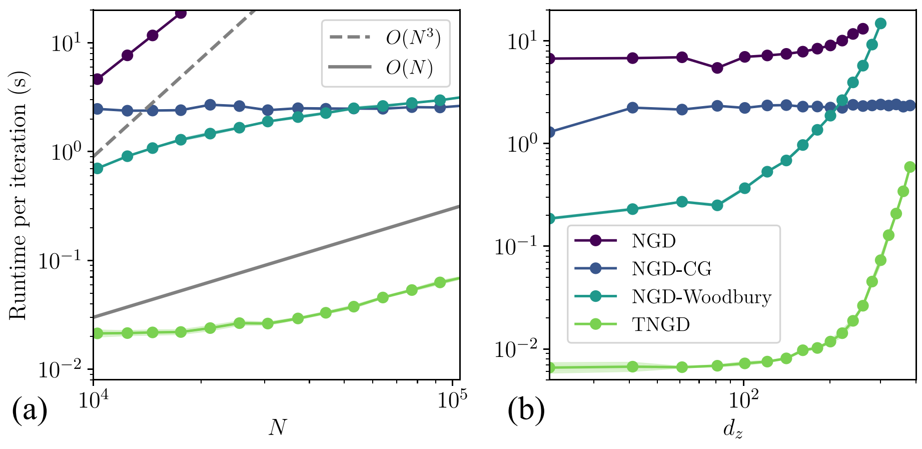

Fig. 2 | Runtime per iteration of second-order optimizers considered in this paper. a The runtimes per iteration are compared for NGD, NGD-CG, NGD-Woodbury, and TNGD (estimated) for various N. Here, the convolutional network we applied to MNIST is used and the dimension of the hidden layer is varied to vary N for fixed $d_z = 20$. **b** The same comparison is shown for various values of d_z. The same network is used and d_z is varied (this also has the effect of varying N, see Supplementary Material). Error bars are displayed as shaded area but are smaller than the data markers.

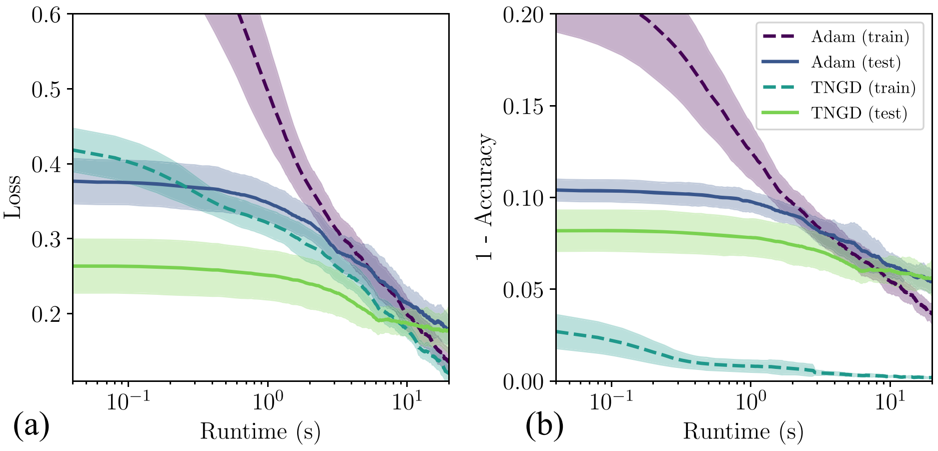

Fig. 3 | Performance comparison of Adam and TNGD (estimated) on MNIST classification. a Training (dashed lines) and test loss (solid lines) for Adam (darker colors) and TNGD (lighter colors) are plotted against runtime (measured for Adam, and estimated for TNGD from the timing model described in the computational complexity section). Shaded areas are standard deviations over five random seeds. Note that Adam includes adaptive averaging of first and second moment estimates with $(\beta_1, \beta_2) = (0.9, 0.999)$, while TNGD does not. **b** 1 − Accuracy for training and test sets.

however, here the batch size is smaller than in other works, indicating that even a noisy GGN matrix improves the optimization.

As mentioned previously, the continuous-time nature of TNGD allows one to interpolate smoothly between first- ($t = 0$) and second- ($t = \infty$) order optimization, with a given optimizer choice (whether the optimizer update rule is that of SGD or that of Adam as described in Alg. 1). In Fig. 4(a), the training loss vs. iterations is shown for various analog dynamics times. These results clearly demonstrate the effect mentioned above, where increasing the analog runtime improves performance continuously until it approaches that of exact NGD for $t \sim 50\tau$ *for time constant of the system* τ. In Fig. 4(b), the same quantity is shown for a fixed analog dynamics time t, and varying delay times t_d. This leads to a quadratic approximation of the objective function that is inaccurate (since the GGN and gradients are calculated for parameters different than the value around which the objective function is approximated). However, this results in an improved performance, even for a small delay time. A likely explanation of this result is that the state of the device retains information about the curvature of the previous quadratic approximation, while being subject to the updated quadratic approximation. This effect propagates across iterations which is reminiscent of momentum.

Language model fine-tuning. In this section, we show how thermodynamic NGD may be applied to language modeling tasks, in more practically relevant settings than MNIST classification. We consider the DistilBert model[32], which we fine-tune on the Stanford question-answering dataset (SQuaD)[33], a common dataset to evaluate model comprehension of technical domains through extractive question-answering. As is commonly performed when fine-tuning, we apply a low-rank adaptation[34] to the model, which reduces its trainable parameters (details about this procedure are in the Supplementary Material) to a manageable amount ($75k$ here) for limited compute resources.

Figure 5 (a) displays a comparison of the training loss for different optimizers. The bare TNGD (as used in the previous section) shows a worse performance than Adam in this setting. However, a hybrid approach, TNGD-Adam, where the natural gradient estimate is used in conjunction with the Adam update rule gives the best performance (this is explained in Methods). One possible explanation for this result is that there are two preconditionings of the gradient for TNGD-Adam: the first comes from the natural gradient, which incorporates curvature information, and the second comes from the Adam update rule, which acts as a signal-noise ratio as explained in ref. 2, which further adjusts the natural gradient values. In Fig. 5(b), we show that the same results as in the previous section apply to TNGD-Adam, where increasing the analog runtime boosts performance. Therefore, the analog runtime in TNGD may be viewed as a resource in this sense, that is, computationally very cheap (as time constants can be engineered to be very small).

Discussion

We first discuss practical considerations and limitations of the proposed hybrid approach. The practical impact of our work relies on the future availability of analog thermodynamic computers, such as a scaled up version of the system in ref. 21. We provide a circuit diagram of a potential thermodynamic computer in the Supplementary Material. Such computers can employ standard electrical components and leverage CMOS-based fabrication infrastructure, and hence are likely straightforward to scale up, although that remains to be demonstrated.

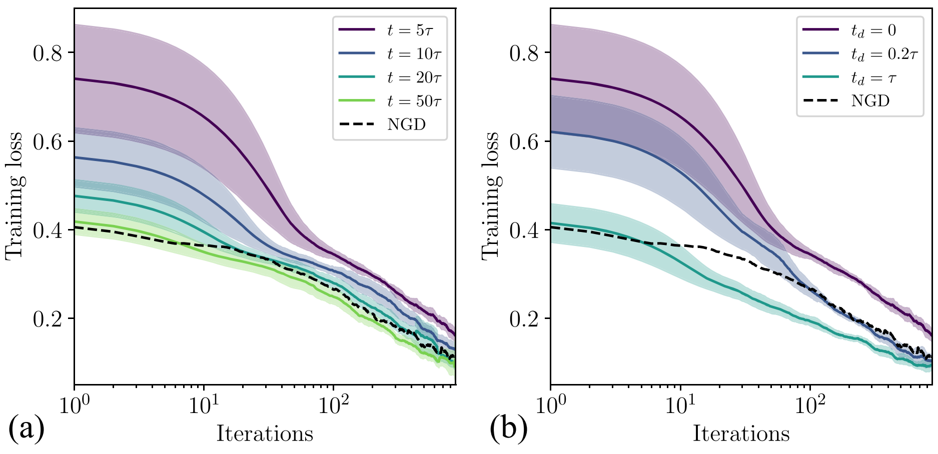

Fig. 4 | Training loss vs. iterations for varying analog dynamics times. a The training loss is shown for NGD (dashed line) and for TNGD with various analog dynamics times t (solid lines). **b** The training loss is shown for NGD (dashed line) and for TNGD with fixed analog dynamics time $t = 5\tau$ and varying delay times t_d (solid lines). The delay appears to have a momentum effect, which can even lead to TNGD outperforming exact NGD for certain analog dynamics and delay times. Shaded areas are standard deviations over five random seeds.

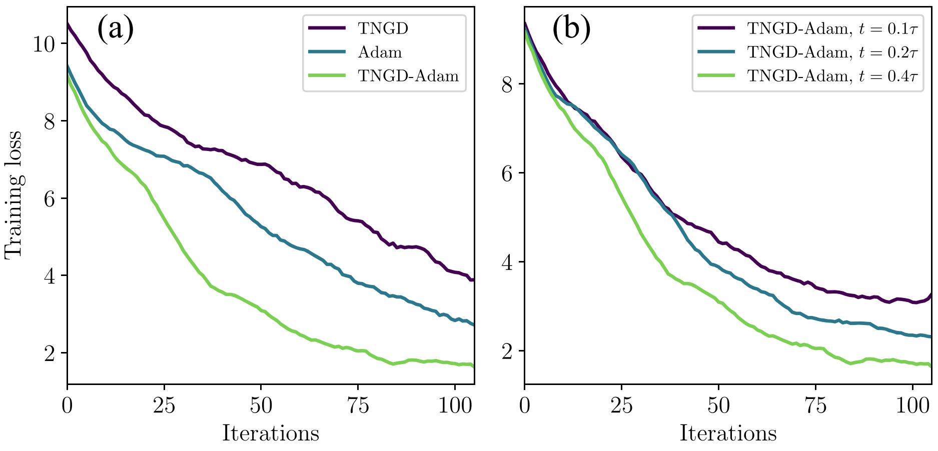

Fig. 5 | Training loss vs. iterations for QA fine-tuning. a Comparison of the performance per iteration of TNGD, Adam, and TNGD-Adam, where the latter uses the natural gradient estimate in conjunction with the Adam update rule with $(\beta_1, \beta_2) = (0, 0)$. **b** Performance of the TNGD-Adam optimizer for various analog dynamics times. Similar to Fig. 4, the performance improves as t grows.

Analog computers are inherently subject to errors. The main sources of such errors in the analog thermodynamic system are (i) limited resolution of electrical components and ADC/DAC conversions, which perturb the matrix inverted in the NGD update; (ii) component tolerances due to the fabrication process; (iii) thermal and other electrical noise sources, such as power fluctuations or external interference; and (iv) numerical inaccuracies in the hybrid digital-analog loop, which may accumulate over iterations, as the accuracy of the solution depends on the resolution of physical components and the analog-to-digital conversions. In the case of analog thermodynamic computers, errors of the type (i) and (ii) manifest as perturbations to the matrix inverted in the NGD update (4), thereby introducing inaccuracies in the natural gradient estimate. These forms of error have been extensively characterized in ref. 22 for the type of circuit presented here, where a theoretically justified averaging technique was also proposed as a mitigation strategy. Moreover, the impact of limited precision on convergence of related second-order methods such as K-FAC has been studied in ref. 35, and found to only significantly affect convergence for resolutions below roughly 8 bits for both the input (coming from component resolution) and the output (coming from ADC resolution). We further note that training-based applications are expected to tolerate precision-related errors, in line with a growing body of work showing that very low-precision inference[36] and training[35] remain effective. A key feature of thermodynamic computers is that stochastic fluctuations in resistor voltages, as described by Eq. C1, do not alter the stationary properties of the system. These fluctuations only increase the variance of voltages, which can be mitigated by extending the averaging time for the natural gradient estimate. The effect of such noise on convergence when using a single sample is shown in the Supplementary Material, where thermal noise has only a slight impact on loss at realistic component and temperature values.

We have numerically tested TNGD for a small subset of potential tasks, such as MNIST classification and DistilBert fine-tuning on the SQuaD dataset, for a small number of epochs. Hence, seeing if the advantage we observe for TNGD also holds for other applications is an important direction. Most prominently, we would be interested in the performance of the hybrid algorithm as the complexity of the model or amount of data increases. However, we have strong confidence that the approach would perform well and it's benefits even amplified in this regime due to the canonical preconditioning provided the dense Fisher information matrix which is backed up with established theory[4,5] combined with the favorable scaling of our algorithm, Table 1.

In summary, this work introduced Thermodynamic Natural Gradient Descent (TNGD), a hybrid digital-analog algorithm that leverages the thermodynamic properties of an analog system to efficiently perform second-order optimization. TNGD greatly reduces the computational overhead typically associated with second-order methods for arbitrary model architectures. Our numerical results on MNIST classification and language model fine-tuning tasks demonstrate that TNGD outperforms state-of-the-art first-order methods, such as Adam, and provides large speedups over other second-order optimizers. This suggests a promising future for second-order methods when integrated with specialized hardware.

Looking forward, our research stimulates further investigation into TNGD, particularly with enhancements such as averaging techniques and moving averages. Extensions to approximate second-order methods such as K-FAC may also be possible. Moreover, the principles of thermodynamic computing could inspire new algorithms for Bayesian filtering. In addition, natural gradient descent techniques can be employed under a variety of learning rules, where TNGD could be exploited[37]. While the current impact of our work relies on the development and availability of large-scale analog thermodynamic computers, the theoretical and empirical advantages presented here underscore the potential of co-designing algorithms and hardware to overcome the limitations of conventional digital approaches.

Methods

Natural gradient descent

We have considered a generic supervised learning setting, where the goal is to minimize an objective function defined as:

$$\ell(\theta) = \frac{1}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}} L(y, f_\theta(x)), \tag{9}$$

where $L(y, f_\theta(x)) \in \mathbb{R}$ is a loss function, $f_\theta(x)$ is the forward function that is parametrized by $\theta \in \mathbb{R}^N$. These functions depend on input data and labels $(x, y) \in \mathcal{D}$, with $\mathcal{D}$ a given training dataset. Viewed through the lens of statistics, minimizing the objective function is analogous to minimizing the Kullback-Leibler (KL) divergence from the target joint distribution $q(x, y)$ to the learned distribution $p(x, y|\theta)$[5]. A straightforward way to optimize $\ell(\theta)$ is to follow the direction of steepest descent, defined by the negative gradient $-\nabla\ell$, defined as:

$$\frac{-\nabla\ell}{||\nabla\ell||} = \lim_{\epsilon\to 0}\frac{1}{\epsilon}\ \arg\ \underset{d:||d||\le\epsilon}{min}\ \ell(\theta + d), \tag{10}$$

with $||\cdot||$ the Euclidean norm. The natural gradient, on the other hand can be defined as the direction of steepest descent with respect to the KL divergence defined as:

$$\mathrm{KL}(p(x,y|\theta+d)||p(x,y|\theta)) = \iint p(x,y|\theta+d)\,\log\left(\frac{p(x,y|\theta+d)}{p(x,y|\theta)}\right)\mathrm{d}x\mathrm{d}y \tag{11}$$

ref. 38. One may then Taylor-expand this divergence as

$$KL(p(x,y|\theta+d)||p(x,y|\theta)) = \frac{1}{2}d^\top F d + O(d^3), \tag{12}$$

where F is the *Fisher information matrix*[5] (or the *Fisher*), defined as:

$$F = \mathbb{E}_{p(x,y|\theta)}[\nabla\log p(x,y|\theta)\nabla\log p(x,y|\theta)^\top]. \tag{13}$$

The natural gradient is then simply defined as

$$\tilde{g} = F^{-1}\nabla\ell(\theta). \tag{14}$$

For the NGD optimizer, the update rule is then given by:

$$\theta_{k+1} = \theta_k - \eta F^{-1}\nabla\ell, \tag{15}$$

with η a learning rate. In practice, computing the Fisher information is not always feasible because one must have access to the density $p(x, y|\theta)$. A quantity that is always possible (and relatively cheap) to compute thanks to auto-differentiation is the empirical Fisher information matrix, defined as:

$$\bar{F} = JJ^\top = \frac{1}{b}\sum_{(x,y)\in\mathcal{S}}\nabla\log p(y|x,\theta)\nabla\log p(y|x,\theta)^\top, \tag{16}$$

where $\log p(y|x,\theta) = -L(y, f_\theta(x))$, $|\mathcal{S}| = b$ is the batch size and $\mathcal{S}\subset\mathcal{D}$. The Jacobian matrix J is defined as

$$J = \frac{1}{\sqrt{b}}[\nabla\log p(y_1|x_1,\theta), \nabla\log p(y_2|x_2,\theta), \dots, \nabla\log p(y_b|x_b,\theta)].$$

Note that the squared gradient appearing in the second moment estimate of the Adam optimizer[2] is the diagonal of the empirical Fisher matrix. Another approximation to the Fisher matrix is the generalized Gauss-Newton (GGN) matrix, defined as:

$$G = J_f H_L J_f^\top = \frac{1}{b}\sum_{(x,y)\in\mathcal{S}} J_f^{(x,y)} H_L^{(x,y)} J_f^{(x,y)\top}, \tag{17}$$

where $J_f^{(x,y)}$ is the Jacobian of $f_\theta(x)$ with respect to θ and $H_L^{(x,y)}$ is the Hessian of $L(y, z)$ with respect to z evaluated at $z = f_\theta(x)$. J_f is a $bd_z\times N$ matrix, and H_L is a $bd_z\times bd_z$ matrix, where d_z is the output dimension of $z = f_\theta(x)$ and N is the number of parameters (N also depends on d_z, where for deep networks it is a weak dependence).

For loss functions of the exponential family (with natural parameter z), the GGN matches the true Fisher matrix[5]. In addition, we have observed better convergence with the GGN than with the empirical Fisher (as in other works such as Refs. 24,39, where better convergence than with the Hessian is also observed). Therefore, we will consider the GGN in what follows. Note that the methods we introduce in this work apply to any second-order optimization algorithm with a positive semi-definite curvature matrix (by curvature matrix, we mean any matrix capturing information about the loss landscape). In particular, it applies most efficiently to matrices constructed as outer products of rectangular matrices (such as the empirical Fisher and the GGN) as explained below.

Fast matrix vector products

The linear system appearing in (3) can be solved using the conjugate gradient (CG) method[24], which will be referred to as NGD-CG in what follows. In fact, when ℓ is parametrized by a neural network, the GGN-vector product Gv involved in the conjugate gradient algorithm may be evaluated in runtime $O(bN)$ thanks to fast Jacobian-vector products[40] (JVPs). This approach also enables one to not explicitly construct the Fisher matrix, thus also avoiding a $O(bd_zN^2)$ runtime cost in computing it and a $O(N^2)$ memory cost in storing it. The efficiency of this approach depends on the number of CG iterations required to obtain good performance. Importantly, fully reaching convergence (which takes in $\sqrt{\kappa}$ steps, with κ the condition number of F) is not required to obtain competitive performance[6,27]. Crucially, due to the sequential nature of the algorithm, the CG iterations cannot be parallelized.

In practice, since reaching convergence is computationally expensive, one generally stops the CG algorithm after a set number of iterations. Because of the way the step size is adapted in CG, we have observed that the solution after k steps x_k is not necessarily closer to the true solution than the initial guess x_0, in particular for ill-conditioned problems, which can make NGD-CG difficult to use.

NGD with the Woodbury identity

In the machine learning setting, it is often the case that $b \ll N$ (and $d_z \ll N$). This means that the curvature matrix is low-rank and the linear system to solve is underdetermined. To mitigate this issue, the Fisher matrix may be dampened as $F + \lambda\mathbb{I}$. In that case, the Woodbury identity may be used to obtain the inverse Fisher vector-product $F^{-1}v$ appearing in the NGD update. We have:

$$F = UV + \lambda\mathbb{I}, \text{with } U = J_f,\ V = H_L J_f^\top \tag{18}$$

$$F^{-1} = \lambda^{-1}\mathbb{I} - \lambda^{-2}U(\mathbb{I} + \lambda^{-1}VU)^{-1}V \qquad \text{(Woodbury)} \tag{19}$$

$$F^{-1}v = \lambda^{-1}v - \lambda^{-2}U(\mathbb{I} + \lambda^{-1}VU)^{-1}Vv \tag{20}$$

This is included in[26], and can be competitive with NGD-CG when the batch size b and output dimension d_z are much smaller than the number of trainable parameters N. Here one must construct the V matrix, which has runtime $O(d_z^2 bN)$ (since H_L is block-diagonal), and invert $(\mathbb{I} + \lambda^{-1} VU)$ which is $O(b^3 d_z^3)$. While the batch size typically remains small, the value of d_z can make this inversion intractable. For example, in many language-model tasks, $d_z \sim O(10^4)$ is the vocabulary size.

TNGD algorithm

In Alg. 1 we provide the steps for the TNGD algorithm. This algorithm may be used in conjunction with various digital optimizers (such as SGD or Adam). The thermodynamic linear solver (TLS) is performed by an analog thermodynamic computer whose physical implementation is described in the Supplementary Material. The TLS takes as inputs the Jacobian $J_{f,k}$, the Hessian H_L, the gradient g_k and an initial point x_0 (that can be reset at each iteration, or not, in which case $t_d > 0$).

Algorithm 1. Thermodynamic Natural Gradient Descent

```
Require: $n > 0$
    Initialize $\theta_0$
    $\tilde{g}_0 \leftarrow \nabla \ell(\theta_0)$
    optimizer $\leftarrow$ SGD($\eta, \beta$) or Adam($\eta, \beta_1, \beta_2$)
    while $k \neq n$ do
        $x_k, y_k \leftarrow$ next batch
        $g_k \leftarrow \nabla\, \ell(\theta_k, x_k, y_k)$
        $\tilde{g}_k \leftarrow TLS(J_f, H_L, b = g_k, x_0 = \tilde{g}_{k-1})$
        $optimizer.update(\theta_k, \tilde{g}_k)$
        $k \leftarrow k + 1$
    end while
```

Data availability

All data reflected in our results was generated in a way that is clear from the text, and the results can be reproduced by simulating the algorithms as they are described. No external data was used. Results from specific simulations are available upon request.

Code availability

The underlying code for this study is not publicly available but may be made available to qualified researchers on reasonable request from the corresponding author.

Received: 13 January 2025; Accepted: 7 December 2025;
Published online: 27 January 2026

References

1. Khan, H. N., Hounshell, D. A. & Fuchs, E. R. Science and research policy at the end of Moore's law. *Nat. Electron.* **1**, 14–21 (2018).
2. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations* http://arxiv.org/abs/1412.6980 (ICLR, 2015).
3. Loshchilov, I. & Hutter, F. Decoupled weight decay regularization. In *Proceedings of the 7th International Conference on Learning Representations* https://openreview.net/forum?id=Bkg6RiCqY7 (ICLR, 2019).
4. Amari, S.-I. Natural gradient works efficiently in learning. *Neural Comput.* **10**, 251–276 (1998).
5. Martens, J. New insights and perspectives on the natural gradient method. *J. Mach. Learn. Res.* **21**, 5776–5851 (2020).
6. Martens, J. & Grosse, R. Optimizing neural networks with KRonecker-factored approximate curvature. In *International Conference On Machine Learning*, 2408–2417 https://proceedings.mlr.press/v37/martens15.html (PMLR, 2015).
7. Lin, W. et al. Structured inverse-free natural gradient: Memory-efficient & numerically-stable kfac for large neural nets. *arXiv preprint arXiv:2312.05705* https://arxiv.org/abs/2312.05705 (2023).
8. Eschenhagen, R., Immer, A., Turner, R. Schneider, F. & Hennig, P. Kronecker-factored approximate curvature for modern neural network architectures. In *Advances in Neural Information Processing Systems*, vol. 36 https://proceedings.neurips.cc/paper_files/paper/2023/file/6a6679e3d5b9f7d5f09cdb79a5fc3fd8-Paper-Conference.pdf (2023).
9. Pauloski, J. G., Zhang, Z., Huang, L., Xu, W. & Foster, I. T. Convolutional neural network training with distributed k-fac. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–12 https://ieeexplore.ieee.org/abstract/document/9355234 (IEEE, 2020).
10. Kim, S., Gokmen, T., Lee, H.-M. & Haensch, W. E. Analog CMOS-based resistive processing unit for deep neural network training. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 422–425 https://ieeexplore.ieee.org/abstract/document/8052950 (IEEE, 2017).
11. Ambrogio, S. et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018).
12. Cristiano, G. et al. Perspective on training fully connected networks with resistive memories: device requirements for multiple conductances of varying significance. *J. Appl. Phys*. **124**, 151901 (2018).
13. Aguirre, F. et al. Hardware implementation of memristor-based artificial neural networks. *Nat. Commun.* **15**, 1974 (2024).
14. Conte, T. et al. Thermodynamic computing. *arXiv preprint arXiv:1911.01968* https://arxiv.org/abs/1911.01968 (2019).
15. Hylton, T. Thermodynamic neural network. *Entropy* **22**, 256 (2020).
16. Ganesh, N. A thermodynamic treatment of intelligent systems. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, 1–4 (ICRC, 2017).
17. Coles, P. J. et al. Thermodynamic AI and the fluctuation frontier. In *2023 IEEE International Conference on Rebooting Computing (ICRC)*, 1–10 (IEEE, 2023).
18. Lipka-Bartosik, P., Perarnau-Llobet, M. & Brunner, N. Thermodynamic computing via autonomous quantum thermal machines. *Sci. Adv.* **10**, eadm8792 (2024).
19. Aifer, M. et al. Thermodynamic linear algebra. *npj Unconv. Comput.* **1**, 13 (2024).
20. Duffield, S., Aifer, M., Crooks, G., Ahle, T. & Coles, P. J. Thermodynamic matrix exponentials and thermodynamic parallelism. *Phys. Rev. Res.* **7**, 013147 (2025).
21. Melanson, D. et al. Thermodynamic computing system for AI applications. *Nat. Commun*. **16** https://doi.org/10.1038/s41467-025-59011-x (2025).
22. Aifer, M. et al. Error mitigation for thermodynamic computing. *arXiv preprint arXiv:2401.16231*https://arxiv.org/abs/2401.16231 (2024).
23. Bottou, L., Curtis, F. E. & Nocedal, J. Optimization methods for large-scale machine learning. *SIAM Rev.* **60**, 223–311 (2018).
24. Martens, J. et al. Deep learning via Hessian-free optimization. In *ICML*, vol. 27, 735–742 https://www.cs.toronto.edu/asamir/cifar/HFO_James.pdf (2010).
25. Martens, J., Ba, J. & Johnson, M. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations* https://openreview.net/pdf?id=HyMTkQZAb (2018).
26. Ren, Y. & Goldfarb, D. Efficient subsampled Gauss-Newton and natural gradient methods for training neural networks. *arXiv preprint arXiv:1906.02353* https://arxiv.org/abs/1906.02353 (2019).
27. Gargiani, M., Zanelli, A., Diehl, M. & Hutter, F. On the promise of the stochastic generalized Gauss-Newton method for training DNNs. *arXiv preprint arXiv:2006.02409* https://arxiv.org/abs/2006.02409 (2020).
28. Shen, Z., Zhang, M., Zhao, H., Yi, S. & Li, H. Efficient attention: Attention with linear complexities. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 3531–3539 https://ieeexplore.ieee.org/abstract/document/9423033 (2021).
29. Katharopoulos, A., Vyas, A., Pappas, N. & Fleuret, F. Transformers are RNNS: Fast autoregressive transformers with linear attention. In

International Conference on Machine Learning, 5156–5165 https://proceedings.mlr.press/v119/katharopoulos20a.html (PMLR, 2020).
30. Wang, S., Li, B. Z., Khabsa, M., Fang, H. & Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* https://arxiv.org/abs/2006.04768 (2020).
31. LeCun, Y. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/ (1998).
32. Sanh, V., Debut, L., Chaumond, J. & Wolf, T. Distilbert, a distilled version of Bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* https://arxiv.org/abs/1910.01108 (2019).
33. Rajpurkar, P., Zhang, J., Lopyrev, K. & Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* https://arxiv.org/abs/1606.05250 (2016).
34. Hu, E. J. et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations* https://openreview.net/forum?id=nZeVKeeFYf9 (ICLR, 2022).
35. Sun, X. et al. Ultra-low precision 4-bit training of deep neural networks. *Adv. Neural Inf. Process. Syst.* **33**, 1796–1807 (2020).
36. Ma, S. et al. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764* https://arxiv.org/abs/2402.17764 (2024).
37. Shoji, L., Suzuki, K. & Kozachkov, L. Is all learning (natural) gradient descent? *arXiv preprint arXiv:2409.16422* (2024).
38. Amari, S.-i. & Nagaoka, H. *Methods of information geometry*, vol. 191 https://bookstore.ams.org/mmono-191 (American Mathematical Soc., 2000).
39. Kunstner, F., Hennig, P. & Balles, L. Limitations of the empirical Fisher approximation for natural gradient descent. *Advances in Neural Information Processing Systems* **32** https://proceedings.neurips.cc/paper/2019/hash/46a558d97954d0692411c861cf78ef79-Abstract.html (2019).
40. Bradbury, J. et al. JAX: composable transformations of Python +NumPy programs http://github.com/google/jax (2018).

Acknowledgements

Not applicable.

Author contributions

K.D. and S.D. hold equal contribution in carrying out the experiments and coming up with mathematical equations and proofs based on S.D.'s original idea. K.D., S.D., M.A. derived the equations in the main text and supplementary material. K.D., S.D, M.A., G.C and P.C. wrote the main text. D.M. designed the hardware implementation and wrote the corresponding supplementary material.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s44335-025-00049-x.

Correspondence and requests for materials should be addressed to Samuel Duffield.