Traditional and Accelerated Gradient Descent for Neural Architecture Search

Nicolás García Trillos[1(✉)], Félix Morales[2], and Javier Morales[3]

[1] University of Wisconsin Madison, Madison, WI 53711, USA
garciatrillo@wisc.edu
[2] Departamento de Ingeniería Informática, Universidad Católica Andrés Bello, Capital District, Caracas 1000, Venezuela
famorales.14@est.ucab.edu.ve
[3] Center for Scientific Computation and Mathematical Modeling (CSCAMM), University of Maryland, College Park, MD 20742, USA
javier.moralesdelgado16@gmail.com

Abstract. In this paper we introduce two algorithms for neural architecture search (NASGD and NASAGD) following the theoretical work by two of the authors [4] which used the geometric structure of optimal transport to introduce the conceptual basis for new notions of traditional and accelerated gradient descent algorithms for the optimization of a function on a semi-discrete space. Our algorithms, which use the network morphism framework introduced in [1] as a baseline, can analyze forty times as many architectures as the hill climbing methods [1,10] while using the same computational resources and time and achieving comparable levels of accuracy. For example, using NASGD on CIFAR-10, our method designs and trains networks with an error rate of 4.06 in only 12 h on a single GPU.

Keywords: Neural networks · Neural architecture search · Gradient flows · Optimal transport · Second order dynamics · Semi-discrete optimization

1 Introduction

Motivated by the success of neural networks in applications such as image recognition and language processing, in recent years practitioners and researchers have devoted great efforts in developing computational methodologies for the

NGT was supported by NSF-DMS 2005797. The work of JM was supported by NSF grants DMS16-13911, RNMS11-07444 (KI-Net) and ONR grant N00014-1812465. The authors are grateful to the CSCAMM, to Prof. P.-E. Jabin and Prof. E. Tadmor for sharing with them the computational resources used to complete this work. Support for this research was provided by the Office of the Vice Chancellor for Research and Graduate Education at the University of Wisconsin-Madison with funding from the Wisconsin Alumni Research Foundation.

F. Nielsen and F. Barbaresco (Eds.): GSI 2021, LNCS 12829, pp. 507–514, 2021.
https://doi.org/10.1007/978-3-030-80209-7_55

automatic design of neural architectures in order to use deep learning methods in further applications. There is an enormous literature on neural architecture search methodologies and some of its applications (see [11] for an overview on the subject), but roughly speaking, most approaches for *neural architecture search* (NAS) found in the literature build on ideas from reinforcement learning [12], evolutionary algorithms [8,9], and hill-climbing strategies based on network morphisms [1,10]. All NAS approaches attempt to address a central difficulty: the high computational burden of training multiple architecture models. Several developments in the design of algorithms, implementation, and computational power have resulted in methodologies that are able to produce neural networks that outperform the best networks designed by humans. Despite all the recent exciting computational developments in NAS, we believe that it is largely of interest to propose sound mathematical frameworks for the design of new computational strategies that can better explore the architecture space and ultimately achieve higher accuracy rates in learning while reducing computational costs.

In this paper we propose two new algorithms for NAS: *neural architecture search gradient descent* **(NASGD)** and *neural architecture search accelerated gradient descent* **(NASAGD)**. These algorithms are based on: 1) the mathematical framework for semi-discrete optimization (deeply rooted in the geometric structure of optimal transport) that two of the authors have introduced and motivated in their theoretical work [4], and 2) the neural architecture search methods originally proposed in [1,10]. We have chosen the *network morphism* framework from [1] because it allows us to illustrate the impact that our mathematical ideas can have on existing NAS algorithms without having to introduce the amount of background that other frameworks like those based on reinforcement learning would require. We emphasize that our high level ideas are not circumscribed to the network morphism framework.

In the morphism framework from [1] an iterative process for NAS is considered. In a first step, the parameters/weights of a collection of architectures are optimized for a *fixed time*, and in a second step the set of architectures are updated by applying network morphisms to the best performing networks in the previous stage; these two steps are repeated until some stopping criterion is reached. In both NASGD and NASAGD we also use the concept of network morphism, but now the time spent in training a given set of networks is *dynamically chosen* by an evolving particle system. In our numerical experiments we observe that our algorithms change architectures much earlier than the fixed amount of time proposed in [1], while achieving error rates of 4.06% for the CIFAR-10 data set trained over 12 h with a single GPU for NASGD, and of 3.96% on the same training data set trained over 1 day with one GPU for NASAGD.

2 Our Algorithms

In this section we introduce our algorithms NASGD and NASAGD. In order to motivate them, for pedagogical purposes we first consider an idealized setting

where we imagine that NAS can be seen as a tensorized semi-discrete optimization problem of the form:

$$\min_{(x,g)\in\mathbb{R}^d\times\mathcal{G}} V(x,g). \tag{1}$$

In the above, it will be useful to think of the g coordinate as an *architecture* and the x coordinate as the *parameters* of that architecture. It will also be useful to think of $\mathcal{G}=(\mathcal{G},K)$ as a finite similarity graph of architectures with K a matrix of positive weights characterizing a *small* neighborhood of a given architecture (later on $\mathcal{G}$ is defined in terms of network morphisms around a given architecture –see Sect. 2.3), and V as a loss function (for concreteness cross-entropy loss) which quantifies how well an architecture with given parameters performs in classifying a given training data set. Working in this ideal setting, in the next two subsections we introduce particle systems that aim at solving (1). These particle systems are inspired by the gradient flow equations derived in [4] that we now discuss.

2.1 First Order Algorithm

The starting point of our discussion is a modification of equation (2.13) in [4] now reading:

$$\begin{aligned}\partial_t f_t(x,g) = \sum_{g'\in\mathcal{G}} &\left[\log f_t(g) + V(x,g) - (\log f_t(g') + V(x,g'))\right] \\ &\cdot K(g,g')\theta_{x,g,g'}(f_t(x,g), f_t(x,g')) + \mathrm{div}_x(f_t(x,g)\nabla_x V(x,g)),\end{aligned} \tag{2}$$

for all $t>0$. In the above, $f_t(x,g)$ must be interpreted as a probability distribution on $\mathbb{R}^d\times\mathcal{G}$ and $f_t(g)$ as the corresponding marginal distribution on g. ∇_x denotes the gradient in $\mathbb{R}^d$ and div_x the divergence operator acting on vector fields on $\mathbb{R}^d$. The first term on the right hand side of (2) is a divergence term on the graph acting on graph vector fields (i.e. real valued functions defined on the set of edges of the graph). The term $\theta_{x,g,g'}(f_t(x,g), f_t(x,g'))$ plays the role of interpolation between the masses located at the points (x,g) and (x,g'), and it provides a simple way to define induced masses on the edges of the graph. With induced masses on the set of edges one can in turn define fluxes along the graph that are in close correspondence with the ones found in the dynamic formulation of optimal transport in the Euclidean space setting (see [6]).

The relevance of the evolution of distributions (2) is that it can be interpreted as a continuous time *steepest descent equation* for the minimization of the energy:

$$\widetilde{\mathcal{E}}(f) := \sum_{g\in\mathcal{G}} \log f(g) f(g) + \sum_g \int_{\mathbb{R}^d} V(x,g) f(x,g) \tag{3}$$

with respect to the geometric structure on the space of probability measures on $\mathbb{R}^d\times\mathcal{G}$ that was discussed in section 2.3 in [4]. Naturally, the choice of different interpolators θ endow the space of measures with a different geometry. In [4] the emphasis was given to choices of θ that give rise to a Riemannian structure

on the space of measures, but alternative choices of θ, like the one made in [2], induce a general *Finslerian* structure instead. In this paper we work with an interpolator inducing a Finslerian structure, and in particular define

$$\theta_{x,g,g'}(s,s') := s\mathbb{1}_{U(x,g,g')>0} + s'\mathbb{1}_{U(x,g,g')<0}, \quad s,s'>0, \tag{4}$$

where $U(x,g,g') := \log f_t(g) + V(x,g) - (\log f_t(g') + V(x,g'))$. We notice that with the entropic term used in (3) we only allow "wandering" in the g coordinate. This term encourages exploration of the architecture space.

We now consider a collection of moving particles on $\mathbb{R}^d \times \mathcal{G}$ whose evolving empirical distribution aims at mimicking the evolution described in (2). Initially the particles have locations (x_i, g_i) $i = 1, \ldots, N$ where we assume that if $g_i = g_j$ then $x_i = x_j$ (see Remark 1 below). For fixed time step $\tau > 0$, particle locations are updated by repeatedly applying the following steps:

- **Step 1: Updating parameters (Training)**: For each particle i with position (x_i, g_i) we update its parameters by setting:

$$x_i^\tau = x_i - \tau \nabla_x V(x_i, g_i).$$

- **Step 2: Moving in the architecture space (Mutation)**: First, for each of the particles i with position (x_i, g_i) we decide to change its g coordinate with probability:

$$\tau \sum_j (\log f(g_j) + V(x_j, g_j)) - (\log f(g_i) + V(x_i, g_i))^- K(g_i, g_j),$$

or 1 if the above number is greater than 1. If we decide to move particle i, we move it to the position of particle j, i.e. (x_j, g_j), with probability p_j:

$$p_j \propto \left[\log f(g_j) + V(x_j, g_j) - (\log f(g_i) + V(x_i, g_i))\right]^- K(g_i, g_j).$$

In the above, $f(g)$ denotes the ratio of particles that are located at g. Additionally, $a^- = \max\{0, -a\}$ denotes the negative part of the quantity a.

Remark 1. Given the assumptions on the initial locations of the particles, throughout all the iterations of Step 1 and Step 2 it is true that if $g_i = g_j$ then $x_i = x_j$. This is convenient from a computational perspective because in this way the number of architectures that need to get trained is equal to the number of nodes in the graph (which we recall should be interpreted as a small local graph) and not to the number of particles in our scheme.

Remark 2. By modifying the energy $\widetilde{\mathcal{E}}(f)$ replacing the entropic term with an energy of the form $\frac{1}{\beta+1}\sum_g (f(g))^{\beta+1}$ for some parameter $\beta > 0$, one can motivate a new particle system where in Step 2 every appearance of $\log f$ is replaced with f^β. The effect of this change is that the resulting particle system moves at a slower rate than the version of the particle system as described in Step 2.

2.2 Second Order Algorithm

Our *second order algorithm* is inspired by the system of equations (2.17) in [4] which now reads:

$$\begin{cases} \partial_t f_t(x,g) + \sum_{g'}(\varphi_t(x,g') - \varphi_t(x,g))K(g,g')\theta_{x,g,g'}(f_t(x,g), f_t(x,g)) \\ \qquad\qquad + \mathrm{div}_x(f_t(x,g)\nabla_x\varphi_t) = 0 \\ \partial_t\varphi_t + \frac{1}{2}|\nabla_x\varphi_t|^2 + \sum_{g'}\left(\varphi_t(x,g) - \varphi_t(x,g')\right)^2 K(g,g')\partial_s\theta_{x,g,g'}(f_t(x,g), f_t(x,g')) \\ \qquad\qquad = -[\gamma\varphi_t(x,g) + \log f_t(g) + V(x,g)], \end{cases} \tag{5}$$

for $t > 0$. We use θ as in (4) except that now we set $U(x,g,g') := \varphi_t(x,g') - \varphi_t(x,g)$. System (5) describes a second order algorithm for the optimization of $\widetilde{\mathcal{E}}$ – see sections 2.4 and 3.3 in [4] for a detailed discussion. Here, the function φ_t is a real valued function over $\mathbb{R}^d \times \mathcal{G}$ that can be interpreted as *momentum* variable. $\gamma \geq 0$ is a friction parameter.

System (5) motivates the following particle system, where now we think that the position of a particle is characterized by the tuple (x_i, g_i, v_i) where $x_i, v_i \in \mathbb{R}^d$, $g_i \in \mathcal{G}$, and in addition we have a potential function $\varphi : \mathcal{G} \to \mathbb{R}$ that also gets updated. Initially, we assume that if $g_i = g_j$ then $x_i = x_j$ and $v_i = v_j$. We also assume that initially φ is identically equal to zero.

We summarize the *second order gradient flow dynamics* as the iterative application of three steps:

- **Step 1: Updating parameters (Training):** For each particle i located at (x_i, g_i, v_i) we update its parameters x_i, v_i by setting

$$x_i^\tau = x_i + \tau v_i, \quad v_i^\tau \quad = v_i - \tau(\gamma v_i + \nabla_x V(x_i, g_i)).$$

- **Step 2: Moving in the architecture space (Mutation):** First, for each of the particles i with position (x_i, g_i, v_i) we decide to move it with probability

$$\tau \sum_j (\varphi(g_i) - \varphi(g_j))^- K(g_i, g_j),$$

  or 1 if the above quantity is greater than 1. Then, if we decided to move the particle i we move it to location of particle j, (x_j, g_j, v_j) with probability p_j

$$p_j \propto (\varphi(g_i) - \varphi(g_j))^- K(g_i, g_j).$$

- **Step 3: Updating momentum on the g coordinate**: We update φ according to:

$$\varphi^\tau(g_i) = \varphi(g_i) - \frac{\tau}{2}|v_i|^2 - \tau\left(\sum_j \left([\varphi(g_i) - \varphi(g_j)]^-\right)^2 K(g_i, g_j)\right)$$
$$- \tau(\gamma\varphi(g_i) + \log f(g_i) + V(x_i, g_i)),$$

  for every particle i. Here, $f(g)$ represents the ratio of particles located at g.

Remark 3. Notice that given the assumption on the initial locations of the particles, throughout all the iterations of Step 1 and Step 2 and Step 3 we make sure that if $g_i = g_j$ then $x_i = x_j$ and $v_i = v_j$.

2.3 NASGD and NASAGD

We are now ready to describe our algorithm NASGD:

1. Load an initial architecture g_0 with initial parameters x_0 and set $r = 0$.
2. Construct a graph $\mathcal{G}_r$ around g_r using the notion of network morphism introduced in [1]. More precisely, we produce n_{neigh} new architectures with associated parameters, each new architecture is constructed by modifying g_r using a *single* network morphism from [1]. Then define $\mathcal{G}_r$ as the set consisting of the loaded n_{neigh} architectures and the architecture g_r. Set the graph weights $K(g, g')$ (for example, setting all weights to one).
3. Put N particles on (x_r, g_r) and put 1 "ghost" particle on each of the remaining architectures in $\mathcal{G}_r$. The architectures for these ghost particles are never updated (to make sure we always have at least one particle in each of the architectures in $\mathcal{G}_r$), but certainly their parameters will.
   Then, run the dynamics discussed in Sect. 2.1 on the graph $\mathcal{G}_r$ (or the modified dynamics see Remark 2, and Appendix of the ArXiv version of this paper [3]) until the node in $\mathcal{G}_r \setminus \{g_r\}$ with the most particles g^{max} has twice as many particles as g_r.
   Set $r = r + 1$. Set $g_r = g^{max}$ and $x_r = x^{max}$, where x^{max} are the parameters of architecture g^{max} at the moment of stopping the particle dynamics.
4. If size of g_r exceeds a prespecified threshold (in terms of number of convolutional layers for example) go to 5. If not go back to 2.
5. Train g_r until convergence.

The algorithm NASAGD is defined similarly with the natural adjustments to account for the momentum variables. Details can be found in sections 2 and 4 from the ArXiv version of this paper [3].

3 Experiments

We used NASGD and NASAGD on the CIFAR-10 data set to obtain two architecture models NASGD1 and NASAGD1 respectively (see Appendix in the ArXiv version of this paper [3]). In the next table we compare the performance of NASGD1 and NASAGD1 against our benchmark architectures NASH2 and NasGraph produced by the methodologies proposed in [1,10].

Numerical experiments				
CIFAR 10	Model	Resources	# params $\times 10^6$	Error
	NASH2	1 GPU, 1 day	19.7	5.2
	NASGraph	1 GPU, 20 h	?	4.96
	NASGD1	1 GPU, 12 h	25.4	4.06
	NASAGD1	1 GPU, 1 day	22.9	3.96

Besides producing better accuracy rates, it is worth highlighting that our algorithms can explore many more architectures (about 40 times more) than in [1] with the same computational resources. We took advantage of this faster exploration and considered positive as well as negative architecture mutations, i.e., mutations that can increase or decrease the number of filters, layers, skip, and dimension of convolutional kernels.

Here are some extra details on the implementation of our algorithms. For further details we refer the reader to the ArXiv version of this work [3]. In a similar way to [1] and [10], we pre-train an initial network g_0 with the structure Conv-MaxPool-Conv-MaxPool-Conv-Softmax for 20 epochs using cosine aliasing that interpolates between 0.5 and 10^{-7}; here Conv is interpreted as Conv+batchnorm+Relu. We use g_0 with parameters x_0 as the initial data for our gradient flow dynamics introduced in Sect. 2.1 for the first-order algorithm NASGD and Sect. 5 for the second-order algorithm NASAGD. During the NASGD and NASAGD algorithms, we use cosine aliasing interpolating the learning rate from λ_{start} to λ_{final} with a restart period of $epochs_{neigh}$. In contrast to the NASH approach from [1], since we initialize new architectures, we do not reset the time step along with the interpolation for the $epochs_{neighs}$ epochs. Our particle system *dynamically* determines the number of initialization. We continue this overall dynamics, resetting the learning rate from λ_{start} to λ_{final} every $epochs_{neigh}$ at most n_{steps} times. We perform several experiments letting the first and second-order gradient flow dynamics run for different lengths of time. Finally, we train the found architectures until convergence.

In the table below, we display the rest of the parameters used to find these models.

Variable	NASH2	NASGraph	NASGD1	NASAGD1
n_{steps}	8	10	0.89	2.54
n_{NM}	5	5	*dynamic*	*dynamic*
n_{neigh}	8	8	8	8
$epoch_{neigh}$	17	16	18	18
λ_{start}	0.05	0.1	0.05	0.05
λ_{final}	0	0	10^{-7}	10^{-7}
Gradient stopping	No	Yes	No	No

Here, n_{steps} denotes the number of restart cycles for the cosine aliasing; n_{NM} is the number of morphism operations applied on a given restart cycle; n_{neigh} is the number of children architectures generated every time the current best model changes; $epoch_{neigh}$ is the number of epochs that go by before the cosine aliasing is restarted; λ_{final} and λ_{start} are the parameters required for SGDR.

4 Conclusions and Discussion

In this work we have proposed novel first and second order gradient descent algorithms for neural architecture search: NASGD and NASAGD. The theoretical

gradient flow structures in the space of probability measures over a semi-discrete space introduced in [4] serve as the primary motivation for our algorithms. Our numerical experiments illustrate the positive effect that our mathematical perspective has on the performance of NAS algorithms.

The methodologies introduced in this paper are part of a first step in a broader program where we envision the use of well defined mathematical structures to motivate new learning algorithms that use neural networks. Although here we have achieved competitive results, we believe that there are still several possible directions for improvement that are worth exploring in the future. Some of these directions include: a further analysis of the choice of hyperparameters for NASGD and NASAGD, the investigation of the synergy that our dynamic perspective may have with reinforcement learning approaches (given that more architectures can be explored with our dynamic approach), the adaptation of our geometric and analytic insights to other NAS paradigms such as parameter sharing [7] and differential architecture search [5].

References

1. Elsken, T., Metzen, J.-H., Hutter, F.: Simple and efficient architecture search for convolutional neural networks. arXiv:1711.04528 (2017)
2. Esposito, A., Patacchini, F.S., Schlichting, A., Slepčev, D.: Nonlocal-interaction equation on graphs: gradient flow structure and continuum limit. arXiv:1912.09834 (2019)
3. García-Trillos, N., Morales, F., Morales, J.: Traditional and accelerated gradient descent for neural architecture search. arXiv:2006.15218 (2020)
4. García-Trillos, N., Morales, J.: Semi-discrete optimization through semi-discrete optimal transport: a framework for neural architecture search. arXiv:2006.15221 (2020)
5. Liu, H., Simonyan, K., Yang, Y.: Darts: differentiable architecture search. arXiv:1806.09055 (2018)
6. Maas, J.: Gradient flows of the entropy for finite Markov chains. J. Funct. Anal. **261**(8), 2250–2292 (2011)
7. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 4095–4104. Stockholmsmässan, Stockholm Sweden, PMLR, 10–15 July 2018
8. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI (2018)
9. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
10. Verma, M., Sinha, K., Goyal, A., Verma, S., Susan, P.: A novel framework for neural architecture search in the hill climbing domain. In: 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pp 1–8 (2019)
11. Yu, T., Zhu, H.: Hyper-parameter optimization: a review of algorithms and applications. arXiv:2003.05689 (2020)
12. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv:1611.01578 (2016)