Efficient Neural Architecture Search via Proximal Iterations

Quanming Yao[1*], **Ju Xu**[3*], **Wei-Wei Tu**[1], **Zhanxing Zhu**[2,3,4†]
[1]4Paradigm Inc, [2]School of Mathematical Sciences, Peking University
[3]Center for Data Science, Peking University, [4]Beijing Institute of Big Data Research (BIBDR)
{yaoquanming, tuweiwei}@4paradigm.com, {xuju, zhanxing.zhu}@pku.edu.cn

Abstract

Neural architecture search (NAS) attracts much research attention because of its ability to identify better architectures than handcrafted ones. Recently, differentiable search methods become the state-of-the-arts on NAS, which can obtain high-performance architectures in several days. However, they still suffer from huge computation costs and inferior performance due to the construction of the supernet. In this paper, we propose an efficient NAS method based on proximal iterations (denoted as NASP). Different from previous works, NASP reformulates the search process as an optimization problem with a discrete constraint on architectures and a regularizer on model complexity. As the new objective is hard to solve, we further propose an efficient algorithm inspired by proximal iterations for optimization. In this way, NASP is not only much faster than existing differentiable search methods, but also can find better architectures and balance the model complexity. Finally, extensive experiments on various tasks demonstrate that NASP can obtain high-performance architectures with more than 10 times speedup over the state-of-the-arts.

1 Introduction

Deep networks have been applied to many applications, where proper architectures are extremely important to ensure good performance. Recently, the neural architecture search (NAS) (Zoph and Le 2017; Baker et al. 2017) has been developed as a promising approach to replace human experts on designing architectures, which can find networks with fewer parameters and better performance (Yao et al. 2018; Hutter, Kotthoff, and Vanschoren 2018). NASNet (Zoph and Le 2017) is the pioneered work along this direction and it models the design of convolutional neural networks (CNNs) as a multi-step decision problem and solves it with reinforcement learning (Sutton and Barto 1998).

However, since the search space is discrete and extremely large, NASNet requires a month with hundreds of GPU to obtain a satisfying architecture. Later, observing the good transferability of networks from small to large ones, NASNet-A (Zoph et al. 2017) proposed to cut the networks into blocks and then the search only needs to be carried within such a block or cell. The identified cell is then used as a building block to assemble large networks. Such two-stage search strategy dramatically reduces the size of the search space, and subsequently leads to the significant speedup of various previous search algorithms (e.g., evolution algorithm (Real et al. 2018), greedy search (Liu et al. 2018), and reinforcement learning (Zhong et al. 2018)).

Although the size of search space is reduced, the search space is still discrete that is generally hard to be efficiently searched (Parikh and Boyd 2013). More recent endeavors focused on how to change the landscape of the search space from a discrete to a differentiable one (Luo et al. 2018; Liu, Simonyan, and Yang 2019; Xie et al. 2019). The benefit of such idea is that a differentiable space enables computation of gradient information, which could speed up the convergence of underneath optimization algorithm. Various techniques have been proposed, e.g., DARTS (Liu, Simonyan, and Yang 2019) smooths design choices with softmax and trains an ensemble of networks; SNAS (Xie et al. 2019) enhances reinforcement learning with a smooth sampling scheme. NAO (Luo et al. 2018) maps the search space into a new differentiable space with an auto-encoder.

Among all these works (Tab. 1), the state-of-the-art is DARTS (Liu, Simonyan, and Yang 2019) as it combines the best of both worlds, i.e., fast gradient descent (differentiable search space) within a cell (small search space). However, its search efficiency and performance of identified architectures are still not satisfying enough. As it maintains a supernet during the search, from the computational perspective, all operations need to be forward and backward propagated during gradient descent while only one operation will be selected. From the perspective of performance, operations typically correlate with each other (Xie et al. 2019), e.g., a 7x7's convolution filter can cover a 3x3 one as a special case. When updating a network's weights, the ensemble constructed by DARTS during the search may lead to inferior architecture being discovered. Moreover, as mentioned in (Xie et al. 2019), DARTS is not complete (Tab. 1), i.e., the final structure needs to be re-identified after the search. This causes a bias between

[*] Q. Yao and J. Xu contribute equally, and the work was performed when J. Xu was an intern in 4Paradigm.

[†] Zhanxing Zhu is the corresponding author.

Table 1: Comparison of the proposed NASP with other state-of-the-art NAS methods on four perspectives of searching: differentiable (denoted as "diff"), cell, complete, and constraint.

	space		complete	complexity control	discrete architectures	search algorithm
	diff	cell				
NASNet-A (Zoph et al. 2017)	×	√	√	√	×	reinforcement learning
AmoebaNet (Real et al. 2018)	×	√	√	√	×	evolution algorithm
SNAS (Xie et al. 2019)	√	√	×	√	√	reinforcement learning
DARTS (Liu, et.al. 2019)	√	√	×	×	×	gradient descent
NASP (proposed)	√	√	√	√	√	proximal algorithm

the searched architecture and the final architecture, and might lead to a decay on the performance of the final architecture.

In this work, we propose NAS with proximal iterations (NASP) to improve the efficiency and performance of existing differentiable search methods. We give a new formulation and optimization algorithm to NAS problem, which allows searching in a differentiable space while keeping discrete architectures. In this way, NASP removes the need of training a supernet, then speeds up the search and leads to better architectures. Our contributions are

- Except for the popularly discussed perspectives of NAS, i.e., search space, completeness, and model complexity, we identify a new and important one, i.e., *constraint on architectures* ("discrete architectures" in Tab. 1), to NAS.
- We formulate NAS as a constrained optimization problem, which keeps the space differentiable but enforces architectures being discrete during the search, i.e., only one of all possible operations to be actually employed during forward and backward propagation. This helps improve searching efficiency and decouple different operations during the training. A regularizer is also introduced into the new objective, which allows control of architectures' size.
- Since such discrete constraint is hard to optimize and simple adaptation of DARTS cannot be applied, we propose a new algorithm derived from the proximal iterations (Parikh and Boyd 2013) for optimization. The closed-form solution to the proximal step with the proposed discrete constraint is new to the optimization literature, and removes the expensive *second-order approximation* required by DARTS. We further provide a theoretical analysis to guarantee convergence of the proposed algorithm.
- Finally, experiments are performed with various benchmark data sets on designing CNN and RNN architectures. Compared with state-of-the-art methods, the proposed NASP is not only much faster (more than ten times speedup over DARTS) but also can discover better architectures. These empirically demonstrate NASP can obtain the state-of-the-art performance on both test accuracy and computation efficiency.

The implementation of NASP is available at: https://github.com/xujinfan/NASP-codes.

2 Related Works

In the sequel, vectors are denoted by lowercase boldface, and matrices by uppercase boldface.

2.1 Proximal Algorithm (PA)

Proximal algorithm (PA) (Parikh and Boyd 2013), is a popular optimization technique in machine learning for handling constrained optimization problem as $\min_{\mathbf{x}} f(\mathbf{x}), \text{ s.t. } \mathbf{x} \in \mathcal{S}$, where f is a smooth objective and $\mathcal{S}$ is a constraint set. The crux of PA is the proximal step:

$$\mathbf{x} = \text{prox}_{\mathcal{S}}(\mathbf{z}) = \arg\min_{\mathbf{y}} {}^1\!/_2 \left\|\mathbf{y} - \mathbf{z}\right\|_2^2 \text{ s.t. } \mathbf{y} \in \mathcal{S}. \quad (1)$$

Closed-form solution for the PA update exists for many constraint sets in (1), such as ℓ_1- and ℓ_2-norm ball (Parikh and Boyd 2013). Then, PA generates a sequence of $\mathbf{x}_t$ by

$$\mathbf{x}_{t+1} = \text{prox}_{\mathcal{S}}(\mathbf{x}_t - \varepsilon \nabla f(\mathbf{x}_t)), \quad (2)$$

where ε is the learning rate. PA guarantees to obtain the critical points of f when $\mathcal{S}$ is a convex constraint, and produces limit points when the proximal step can be exactly computed (Yao et al. 2017). Due to its nice theoretical guarantee and good empirical performance, it has been applied to many deep learning problems, e.g., network binarization (Bai, Wang, and Liberty 2018).

Another variant of PA with lazy proximal step (Xiao 2010) maintains two copies of $\mathbf{x}$, i.e.,

$$\bar{\mathbf{x}}_t = \text{prox}_{\mathcal{S}}(\mathbf{x}_t), \quad \mathbf{x}_{t+1} = \mathbf{x}_t - \varepsilon \nabla f(\bar{\mathbf{x}}_t), \quad (3)$$

which is also popularly used in deep learning for network quantization (Courbariaux, Bengio, and David 2015; Hou, Yao, and Kwok 2017). It does not have convergence guarantee in the nonconvex case, but empirically performs well on network quantization tasks. Finally, neither (2) nor (3) have been introduced into NAS.

2.2 Differentiable Architecture Search (DARTS)

DARTS (Liu, Simonyan, and Yang 2019) searchs architecture by cells (Fig. 1(a)), which is a directed acyclic graph consisting of an ordered sequence of N nodes, and it has two input nodes and a single output node (Zoph et al. 2017). Within a cell, each node $x(i)$ is a latent representation and each directed edge (i, j) is associated with some operations $O(i, j)$ that transforms $x(i)$ to $x(j)$. Thus, each intermediate node is computed using all of its predecessors, i.e., $x^{(j)} = \sum_{i<j} O^{(i,j)}(x^{(i)})$ as in Fig. 1(a). However, such search space is discrete. DARTS uses softmax relaxation to make discrete choices into smooth ones (Fig. 1(b)), i.e., each $O^{(i,j)}$ is replaced by $\bar{O}^{(i,j)}$ as

$$\bar{O}^{(i,j)}(x^{(i)}) = {}^1\!/_C \sum\nolimits_{m=1}^{|\mathcal{O}|} \exp(a_m^{(i,j)}) \mathcal{O}_m(x^{(j)}), \quad (4)$$

where $C = \sum_{n=1}^{|\mathcal{O}|} \exp(a_n^{(i,j)})$ is a normalization term, $\mathcal{O}_m$ denotes the m-th operation in search space $\mathcal{O}$. Thus, the choices of operations for an edge (i, j) is replaced by a real vector $\mathbf{a}^{(i,j)} = [a_k^{(i,j)}] \in \mathbb{R}^{|\mathcal{O}|}$, and all choices in a cell can be represented in a matrix $\mathbf{A} = [\mathbf{a}^{(i,j)}]$ (see Fig. 1(d)).

With such a differentiable relaxation, the search problem in DARTS is formulated as

$$\min_{\mathbf{A}} \mathcal{L}_{\text{val}} \left(w^*, \mathbf{A} \right), \text{ s.t. } w^* = \arg\min_{w} \mathcal{L}_{\text{train}} \left(w, \mathbf{A} \right), \quad (5)$$

where $\mathcal{L}_{\text{val}}$ (resp. $\mathcal{L}_{\text{train}}$) is the loss on validation (resp. training) set, and gradient descent is used for the optimization. Let the gradient w.r.t. $\mathbf{A}$ is

$$\begin{aligned} \nabla_{\mathbf{A}} \mathcal{L}_{\text{val}} \left(w, \mathbf{A} \right) = & \nabla_{\mathbf{A}} \mathcal{L}_{\text{val}} \left(\bar{w}(\mathbf{A}), \mathbf{A} \right) \\ & - \varepsilon \nabla^2_{\mathbf{A}, w} \mathcal{L}_{\text{train}}(w, \mathbf{A}) \nabla_{\mathbf{A}} \mathcal{L}_{\text{val}}(\bar{w}, \mathbf{A}), \quad (6) \end{aligned}$$

where $\varepsilon > 0$ is the step-size and a second order derivative, i.e., $\nabla^2_{2,1}(\cdot)$ is involved. However, the evaluation of the second order term is extremely expensive, which requires two extra computations of gradient w.r.t. w and two forward passes of $\mathbf{A}$. Finally, a final architecture $\bar{\mathbf{A}}$ needs to be discretized from the relaxed $\mathbf{A}$ (see Alg.1).

Algorithm 1 Differentiable architecture search (DARTS) (Liu, Simonyan, and Yang 2019).

```
Create a mixture operation $\bar{O}^{(i,j)}$ parametrized by (4);
while not converged do
    Update $\mathbf{A}^t$ by (6);
    // with second-order approximation;
    Update $w_t$ by $\nabla_{w_t} \mathcal{L}_{\text{train}}(w_t, \mathbf{A}^{t+1})$ using back-propagation;
    // with all operations;
end while
Drive the discrete architecture $\bar{\mathbf{A}}$ from continuous $\mathbf{A}$;
// not complete;
return final architecture $\bar{\mathbf{A}}$.
```

Due to the differentiable relaxation in (4), an ensemble of operations (i.e., a supernet) are maintained and all operations in the search space need to be forward and backward-propagated when updating w; the evaluation of the second order term in (6) is very expensive known as a computation bottleneck of DARTS (Xie et al. 2019; Noy et al. 2019). Besides, the performance obtained from DARTS is also not as good as desired. Due to the possible correlations among operations and the need of deriving a new architecture after the search (i.e., lack of completeness) (Xie et al. 2019). Finally, the objective (5) in DARTS does not consider model complexity, which means DARTS cannot control the model size of the final architectures.

3 Our Approach: NASP

As introduced in Sec.2.2, DARTS is a state-of-the-art NAS method, however, it has three significant limitations:

a). *search efficiency*: the supernet resulting obtained from softmax trick in (4) is expensive to train;

b). *architecture performance*: correlations exist in operations, which can lead to inferior architectures.

c). *model complexity*: depending on applications, we may also want to trade accuracy for smaller models; however, this is not allowed in DARTS.

Recall that in earlier works of NAS (see Tab. 1), e.g., NASNet (Baker et al. 2017; Zoph and Le 2017) and GeNet (Xie and Yuille 2017), architectures are discrete when updating networks' parameters. Such discretization naturally avoids the problem of completeness and correlations among operations compared with DARTS. Thus, *can we search in a differentiable space but keep discrete architectures when updating network's parameters*? This motivates us to formulate a new search objective for NAS (Sec.3.1), and propose a new algorithm for optimization (Sec.3.2).

3.1 Search Objective

As NAS can be seen as a black-box optimization problem (Yao et al. 2018; Hutter, Kotthoff, and Vanschoren 2018), here, we bring the wisdom of constraint optimization to deal with the NAS problem.

Discrete constraint Specifically, we keep $\mathbf{A}$ being continuous, which allows the usage of gradient descent, but constrain the values of $\mathbf{A}$ to be discrete ones. Thus, we propose to use the following relaxation instead of (4) on architectures:

$$\bar{O}^{(i,j)}(x^{(i)}) = \sum\nolimits_{k=1}^{|\mathcal{O}|} a_k^{(i,j)} \mathcal{O}_k(x^{(j)}), \text{ s.t. } \mathbf{a}^{(i,j)} \in \mathcal{C}, \quad (7)$$

where the constraint set is defined as $\mathcal{C} = \{\mathbf{a} \mid \|\mathbf{a}\|_0 = 1, \text{and } 0 \le a_k \le 1\}$. While $\mathbf{a}^{(i,j)}$ is continuous, the constraint $\mathcal{C}$ keeps its choices to be discrete, and there is one operation actually activated for each edge during training network parameter w as illustrated in Fig. 1(c).

Regularization on model complexity Besides, in the search space of NAS, different operations have distinct number of parameters. For example, the parameter number of "sep_conv_7x7" is ten times that of operation "conv_1x1". Thus, we may also want to regularize model parameters to trade-off between accuracy and model complexity (Cai, Zhu, and Han 2019; Xie et al. 2019).

Recall that, one column in $\mathbf{A}$ denotes one possible operation (Fig. 1(d)), and whether one operation will be selected depending on its value $\mathbf{a}^{(i,j)}$ (a row in $\mathbf{A}$). Thus, if we suppress the value of a specific column in $\mathbf{A}$, its operation will be less likely to be selected in Alg.2, due to the proximal step on $\mathcal{C}_1$. These motivate us to introduce a regularizer $\mathcal{R}(\mathbf{A})$ as

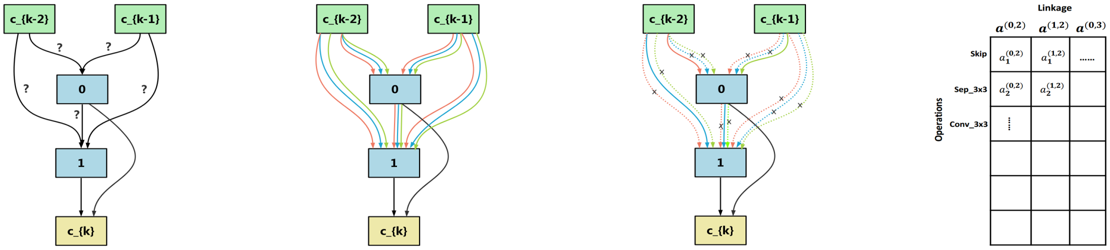
$$\mathcal{R}(\mathbf{A}) = \sum\nolimits_{k=1}^{|\mathcal{O}|} {}^{p_k}\!/_{\bar{p}} \left\| \dot{\mathbf{a}}_k \right\|_2^2, \quad (8)$$

where $\dot{\mathbf{a}}_k$ is the kth column in $\mathbf{A}$, the parameter number with the ith operation is p_i, and $\bar{p} = \sum_{i=1}^{|\mathcal{O}|} p_i$.

Search objective Finally, the NAS problem, with our new relaxation (7) and regularization (8), becomes

$$\min_{\mathbf{A}} \ \mathcal{F}\left(w^*, \mathbf{A}\right), \text{s.t.} \begin{cases} w^* = \arg\min_{w} \mathcal{L}_{\text{train}}\left(w, \mathbf{A}\right) \\ \mathbf{a}^{(i,j)} \in \mathcal{C} \end{cases}, \quad (9)$$

where $\mathcal{F}(w, \mathbf{A}) = \mathcal{L}_{\text{val}}\left(w, \mathbf{A}\right) + \eta \mathcal{R}(\mathbf{A})$ with $\eta \ge 0$ balancing between the complexity and the accuracy, and a larger η leads to smaller architectures.



(a) Cell in NAS. (b) DARTS. (c) NASP. (d) Parameter $\mathbf{A}$.

Figure 1: Comparison of computation graph in a cell between DARTS (Fig. 1(b)) and NASP (Fig. 1(c)). Three operations are considered, DARTS needs to forward and backward propagate along all operations for updating w, while NASP only requires computing along current selected one. The architecture parameters $a_k^{(i,j)}$ can be arranged into a matrix form (Fig. 1(d)).

Remark 1. *Literally, learning with a discrete constraint has only been explored with parameters, e.g., deep networks compression with binary weights (Courbariaux, Bengio, and David 2015), and gradient quantization (Alistarh et al. 2017), but not in hyper-parameter or architecture optimization. Meanwhile, other constraints have been considered in NAS, e.g., memory cost and latency (Tan et al. 2018; Cai, Zhu, and Han 2019). We are the first to introduce searched constraints on architecture into NAS (Tab. 1).*

3.2 Search Algorithm

Solving the new NAS objective (9) here is not trivial. Due to the extra constraint and regularizer, neither simple adaptation of DARTS nor PA can be applied. In the sequel, we propose a new variant of PA algorithm for efficient optimization.

Failure of existing algorithms A direct solution would be PA mentioned in Sec.2.1, then architecture $\mathbf{A}_{t+1}$ can be either updated by (2), i.e.,

$$\mathbf{A}_{t+1} = \text{prox}_{\mathcal{C}}(\mathbf{A}_t - \varepsilon \nabla_{\bar{\mathbf{A}}_t} \mathcal{F}(w(\mathbf{A}_t), \mathbf{A}_t)), \tag{10}$$

or updated by lazy proximal step (3), i.e.,

$$\begin{aligned} \bar{\mathbf{A}}_t &= \text{prox}_{\mathcal{C}}(\mathbf{A}_t), \\ \mathbf{A}_{t+1} &= \mathbf{A}_t - \varepsilon \nabla_{\bar{\mathbf{A}}_t} \mathcal{F}(w(\bar{\mathbf{A}}_t), \bar{\mathbf{A}}_t), \end{aligned} \tag{11}$$

where the gradient can be evaluated by (6) and computation of second-order approximation is still required. Let $\mathcal{C}_1 = \{\mathbf{a} \,|\, \|\mathbf{a}\|_0 = 1\}$ and $\mathcal{C}_2 = \{\mathbf{a} \,|\, 0 \le a_k \le 1\}$, i.e., $\mathcal{C} = \mathcal{C}_1 \cap \mathcal{C}_2$. The closed-form solution on proximal step is offered in Proposition 1 (Proofs in Appendix A.1).

Proposition 1. $\text{prox}_{\mathcal{C}}(\mathbf{a}) = \text{prox}_{\mathcal{C}_1}(\text{prox}_{\mathcal{C}_2}(\mathbf{a}))$.

However, solving (9) is not easy. Due to the discrete nature of the constraint set, proximal iteration (10) is hard to obtain a good solution (Courbariaux, Bengio, and David 2015). Besides, while (3) empirically leads to better performance than (2) in binary networks (Courbariaux, Bengio, and David 2015; Hou, Yao, and Kwok 2017; Bai, Wang, and Liberty 2018), lazy-update (11) will not success here neither. The reason is that, as in DARTS (Liu, Simonyan, and Yang 2019), $\mathbf{A}_t$ is naturally in range $[0, 1]$ but (11) can not guarantee that. This in turn will bring negative impact on the searching performance.

Proposed algorithm Instead, motivated by Proposition 1, we keep $\mathbf{A}$ to be optimized as continuous variables but constrained by $\mathcal{C}_2$. Similar box-constraints have been explored in sparse coding and non-negative matrix factorization (Lee and Seung 1999), which help to improve the discriminative ability of learned factors. Here, as demonstrated in experiments, it helps to identify better architectures. Then, we also introduce another discrete $\bar{\mathbf{A}}$ constrained by $\mathcal{C}_1$ derived from $\mathbf{A}$ during iterating. Note that, it is easy to see $\bar{\mathbf{A}}_t \in \mathcal{C}$ is guaranteed. The proposed procedure is described in Alg.2.

Algorithm 2 NASP: Efficient Neural Architecture Search with Proximal Iterations.

```
1: Create a mixture operation $\bar{O}^{(i,j)}$ parametrized by (7);
2: while not converged do
3:     Get discrete architectures: $\bar{\mathbf{a}}_t^{(i,j)} = \text{prox}_{\mathcal{C}_1}(\mathbf{a}_t^{(i,j)})$;
4:     Update $\mathbf{A}_{t+1} = \text{prox}_{\mathcal{C}_2}(\mathbf{A}_t - \nabla_{\bar{\mathbf{A}}_t} \mathcal{F}(w_t, \bar{\mathbf{A}}_t))$;
       // no second-order approximation
5:     Refine discrete architectures: $\bar{\mathbf{a}}_{t+1}^{(i,j)} = \text{prox}_{\mathcal{C}_1}(\mathbf{a}_{t+1}^{(i,j)})$;
6:     Update $w_t$ by $\nabla_{w_t} \mathcal{L}_{\text{train}}(w_t, \bar{\mathbf{A}}^{t+1})$ using back-propagation;
       // with the selected operations
7: end while
8: return Searched architecture $\bar{\mathbf{A}}_t$.
```

Compared with DARTS, NASP also alternatively updates architecture $\mathbf{A}$ (step 4) and network parameters w (step 6). However, note that $\mathbf{A}$ is discretized at step 3 and 5. Specifically, in step 3, discretized version of architectures are more stable than the continuous ones in DARTS, as it is less likely for subsequent updates in w to change $\bar{\mathbf{A}}$. Thus, we can take w_t (step 4) as a constant w.r.t. $\bar{\mathbf{A}}$, which helps us remove the second order approximation in (6) and significantly speed up architectures updates. In step 5, network weights need only to be propagated with the selected operation. This helps to reduce models' training time and decouples operations for training networks. Finally, we do not need an extra step to discretize architecture from a continuous one like DARTS, since a discrete architecture is already maintained during the search. This helps us to reduce the gap between the search and fine-tuning, which leads to better architectures being identified.

Theoretical analysis Finally, unlike DARTS and PA with lazy-updates, the convergence of the proposed NASP can be

Table 2: Classification errors of NASP and state-of-the-art image classifiers on CIFAR-10.

Method	Test Error (%)	Para (M)	Time (GPU days)
DenseNet-BC (Huang et al. 2017)	3.46	25.6	—
NASNet-A + cutout (Zoph et al. 2017)	2.65	3.3	1800
AmoebaNet + cutout (Real et al. 2018)	2.55±0.05	2.8	3150
PNAS (Liu et al. 2018)	3.41±0.09	3.2	225
ENAS (Pham et al. 2018)	2.89	4.6	0.5
Random search + cutout (Liu, Simonyan, and Yang 2019)	3.29±0.15	3.2	4
DARTS (1st-order) + cutout (Liu, Simonyan, and Yang 2019)	3.00±0.14	3.3	1.5
DARTS (2nd-order) + cutout	2.76±0.09	3.3	4
SNAS (large complexity) + cutout (Xie et al. 2019)	2.98	2.9	1.5
SNAS (middle complexity) + cutout	2.85±0.02	2.8	1.5
SNAS (small complexity) + cutout	3.10±0.04	2.3	1.5
NASP (7 operations) + cutout	2.83±0.09	3.3	**0.1**
NASP (12 operations) + cutout	**2.44±0.04**	7.4	0.2

guaranteed in Theorem 2. The proof is in Appendix A.2.

Theorem 2. *Let* $\max \mathcal{F}(w, \mathbf{A}) < \infty$ *and* $\mathcal{F}$ *be differentiable, then the sequence* $\{\mathbf{A}^t\}$ *generated by Alg.2 has limit points.*

Note that, previous analysis cannot be applied. As the algorithm steps are different from all previous works, i.e., (Hou, Yao, and Kwok 2017; Bai, Wang, and Liberty 2018), and it is the first time that PA is introduced into NAS. While two assumptions are made in Theorem 2, smoothness of $\mathcal{F}$ can be satisfied using proper loss functions, e.g., the cross-entropy in this paper, and the second assumption can empirically hold in our experiments.

4 Experiments

Here, we perform experiments with searching CNN and RNN structures. Four datasets, i.e., CIFAR-10, ImageNet, PTB, WT2 will be utilized in our experiments (see Appendix B.1).

4.1 Architecture Search for CNN

Searching Cells on CIFAR-10 Same as (Zoph and Le 2017; Zoph et al. 2017; Liu, Simonyan, and Yang 2019; Xie et al. 2019; Luo et al. 2018), we search architectures on CIFAR-10 ((Krizhevsky 2009)). Following (Liu, Simonyan, and Yang 2019), the convolutional cell consists of $N = 7$ nodes, and the network is obtained by stacking cells for 8 times; in the search process, we train a small network stacked by 8 cells with 50 epochs (see Appendix B.2). Two different search spaces are considered here. The first one is the same as DARTS and contains 7 operations. The second one is larger, which contains 12 operations (see Appendix B.3). Besides, our search space for normal cell and reduction cell is different. For normal cell, the search space only consists of identity and convolutional operations; for reduction cell, the search space only consists of identity and pooling operations.

Results compared with state-of-the-art NAS methods can be found in Tab. 2, the searched cells are in Appendix C.2. Note that ProxlessNAS (Cai, Zhu, and Han 2019), Mnasnet (Tan et al. 2018), and Single Path One-Shot (Guo et al. 2019) are not compared as their codes are not available and they focus on NAS for mobile devices; GeNet (Xie and Yuille 2017) is not compared, as its performance is much worse than ResNet. Note that we remove the extra data augmentation for ASAP except cutout for a fair comparison. We can see that when in the same space (with 7 operations), NASP has comparable performance with DARTS (2nd-order) and is much better than DARTS (1st-order). Then, in the larger space (with 12 operations), NASP is still much faster than DARTS, with much lower test error than others. Note that, NASP on the larger space also has larger models, as will be detailed in Sec.4.3, this is because NASP can find operations giving lower test error, while others cannot.

Regularization on model complexity In above experiments, we have set $\eta = 0$ for (9). Here, we vary η and the results on CIFAR-10 are demonstrated in Fig.2. We can see that the model size gets smaller with larger η.

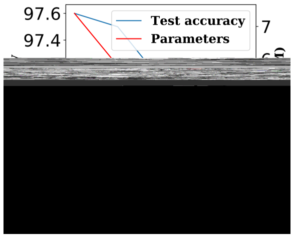


Figure 2: Impact of penalty.

Transfering to ImageNet In order to explore the transferability of our searched cells on ImageNet, we stack the searched cells for 14 times. The experiment results can be seen in Tab. 4. Notably, NASP can achieve competitive test error with the state-of-the-art methods.

4.2 Architecture Search for RNN

Searching cells on PTB Following the setting of DARTS (Liu, Simonyan, and Yang 2019), the recurrent cell consists of $N = 12$ nodes; the first intermediate node is obtained by linearly transforming the two input nodes, adding up the results and then passing through a tanh activation function; then the results of the first intermediate node should be transformed by an activation function. The activation functions utilized are tanh, relu, sigmoid and identity. In the search process, we train a small network with sequence length 35 for 50 epochs. To evaluate the performance of searched cells on PTB, a single-layer recurrent network with the discovered

Table 3: Comparison with state-of-the-art language models on PTB (lower perplexity is better).

Architecture	Perplexity (%)		Params	Time
	valid	test	(M)	(GPU days)
NAS (Zoph and Le 2017)	-	64.0	25	10,000
ENAS (Pham et al. 2018)	68.3	63.1	24	0.5
Random search (Liu, Simonyan, and Yang 2019)	61.8	59.4	**23**	2
DARTS (1st-order) (Liu, Simonyan, and Yang 2019)	60.2	57.6	**23**	0.5
DARTS (2nd-order)	**59.7**	56.4	**23**	1
NASP	59.9	**57.3**	**23**	**0.1**

Table 4: Classification accuracy of NASP and state-of-the-art image classifiers on ImageNet.

Architecture	Test Error (%)		Params
	top1	top5	(M)
Inception-v1 (Szegedy et al. 2015)	30.2	10.1	6.6
MobileNet (Howard et al. 2017)	29.4	10.5	**4.2**
ShuffleNet 2 (Ma et al. 2018)	25.1	10.1	∼5
NASNet-A (Zoph et al. 2017)	26.8	8.4	5.3
AmoebaNet (Real et al. 2018)	**24.3**	**7.6**	6.4
PNAS (Liu et al. 2018)	25.8	8.1	5.1
DARTS (2nd-order)	26.9	9.0	4.9
SNAS (middle complexity)	27.3	9.2	4.3
NASP (7 operations)	27.2	9.1	4.6
NASP (12 operations)	26.3	8.6	9.5

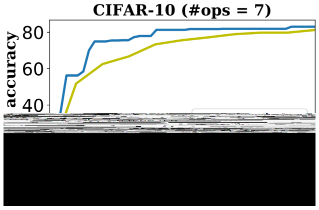
cell is trained for at most 8000 epochs until convergence with batch size 64. Results can be seen in Tab. 3, and searched cells are in Appendix C.2. Again, we can see DARTS's 2nd-order is much slower than 1st-order, and NASP can be not only much faster than DARTS but also achieve comparable test performance with other state-of-the-art methods.

Transferring to Wiki-Text2 Following (Liu, Simonyan, and Yang 2019), we test the transferable ability of RNN's cell with WikiText-2 (WT2) (Pham et al. 2018) dataset. We train a single-layer recurrent network with the searched cells on PTB for at most 8000 epochs. Results can be found in Tab. 7. Unlike previous case with ImageNet, performance obtained from NAS methods are not better than human designed ones. This is due to WT2 is harder to be transferred, which is also observed in (Liu, Simonyan, and Yang 2019).

4.3 Ablation Study

Comparison with DARTS In Sec.4.1, we have shown an overall comparison between DARTS and NASP. Here, we show detailed comparisons on updating network's parameter (i.e., w) and architectures (i.e., $\mathbf{A}$). Timing results and searched performance are in Tab. 5. First, NASP removes much computational cost, as no $2nd$-order approximation of $\mathbf{A}$ and propagating w with selected operations. This clearly justifies our motivation in Sec.3.1. Second, the discretized $\bar{\mathbf{A}}$ helps to decouple operations on updating w, this helps NASP find better operations under larger search space.

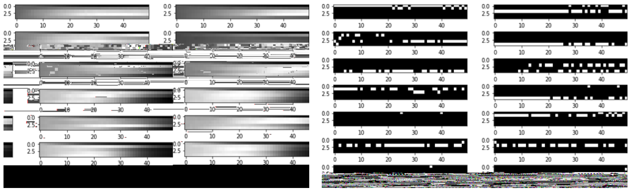
We conduct experiments to compare the search time and validation accuracy in Fig. 3(a)-(b). We can see that in the same search time, our NASP obtains higher accuracy while our NASP cost less time in the same accuracy. This further verifies the efficiency of NASP over DARTS.



(a) #ops = 12. (b) #ops = 7.

Figure 3: Comparison of NASP and DARTS on convergence.

Finally, we illustrate why the second order approximation is a need for DARTS but not for NASP. Recall that, as in Sec.2.2, as $\mathbf{A}$ continuously changes during iteration second order approximation is to better capture w's impact for $\mathbf{A}$. Then, in Sec.3.2, we argue that, since $\bar{\mathbf{A}}$ is discrete, w's impact will not lead to frequent changes in $\bar{\mathbf{A}}$. This removes the needs of capturing future dynamics using the second order approximation. We plot $\mathbf{A}$ for DARTS and $\bar{\mathbf{A}}$ for NSAP in Fig. 4. In Fig. 4, the x-axis represents the training epochs while the y-axis represents the operations (there are five operations selected in our figure). There are 14 connections between nodes, so there are 14 subfigures in both Fig. 4(a) and 4(b). Indeed, $\bar{\mathbf{A}}$ is more stable than $\mathbf{A}$ in DARTS, which verifies the correctness of our motivation.



(a) DARTS. (b) NASP.

Figure 4: Comparison on changes of architecture parameters between DARTS and NASP.

Comparison with standard PA Finally, we demonstrate the needs of our designs in Sec.3.2 for NASP. CIFAR-10 with small search space is used here. Three algorithms are compared: 1). PA (standard), given by (10); 2). PA (lazy-update), given by (11); and 3) NASP. Results are in Fig. 5(a) and Fig. 5(b). First, good performance cannot be obtained from a direct proximal step, which is due to the discrete constraint. Same observation is also previous made for binary

4184090) and Beijing Academy of Artificial Intelligence (BAAI).

References

Akimoto, Y.; Shirakawa, S.; Yoshinari, N.; Uchida, K.; Saito, S.; and Nishida, K. 2019. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *ICML*, 171–180.

Alistarh, D.; Grubic, D.; Li, J.; Tomioka, R.; and Vojnovic, M. 2017. QSGD: Communication-efficient sgd via gradient quantization and encoding. In *NeurIPS*, 1709–1720.

Amari, S. 1998. Natural gradient works efficiently in learning. *Neural Computation* 10(2):251–276.

Bai, Y.; Wang, Y.-X.; and Liberty, E. 2018. Proxquant: Quantized neural networks via proximal operators. In *ICLR*.

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2017. Designing neural network architectures using reinforcement learning. In *ICLR*.

Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NeurIPS*, 3123–3131.

Devries, T., and Taylor, G. 2017. Improved regularization of convolutional neural networks with cutout. Technical report, arXiv:1708.04552.

Dong, X., and Yang, Y. 2019. Searching for a robust neural architecture in four GPU hours. In *CVPR*, 1761–1770.

Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; and Sun, J. 2019. Single path one-shot neural architecture search with uniform sampling. Technical report, Arvix.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Hou, L.; Yao, Q.; and Kwok, J. 2017. Loss-aware binarization of deep networks. In *ICLR*.

Howard, A.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CVPR*.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. 2017. Densely connected convolutional networks. In *CVPR*, 4700–4708.

Hutter, F.; Kotthoff, L.; and Vanschoren, J., eds. 2018. *Automated Machine Learning: Methods, Systems, Challenges*. Springer.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. In *ICLR*.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Le, Y., and Yang, X. 2015. Tiny imagenet visual recognition challenge. *CS 231N*.

Lee, D., and Seung, S. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature* 401:788–791.

Liu, C.; Zoph, B.; Shlens, J.; Hua, W.; Li, L.; Li, F.-F.; Yuille, A.; Huang, J.; and Murphy, K. 2018. Progressive neural architecture search. In *ECCV*.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable architecture search. In *ICLR*.

Luo, R.; Tian, F.; Qin, T.; Chen, E.; and Liu, T.-Y. 2018. Neural architecture optimization. In *NeurIPS*.

Ma, N.; Zhang, X.; Zheng, H.; and Sun, J. 2018. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. *ECCV* 122–138.

Noy, A.; Nayman, N.; Ridnik, T.; Zamir, N.; Doveh, S.; Friedman, I.; Giryes, R.; and Zelnik-Manor, L. 2019. ASAP: Architecture search, anneal and prune. Technical report, arXiv preprint arXiv:1904.04123.

Parikh, N., and Boyd, S. 2013. Proximal algorithms. *Foundations and Trends in Optimization* 1(3):123–231.

Pham, H.; Guan, M.; Zoph, B.; Le, Q.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. Technical report, arXiv preprint.

Real, E.; Aggarwal, A.; Huang, T.; and Le, Q. 2018. Regularized evolution for image classifier architecture search. *arXiv*.

Sutton, R., and Barto, A. 1998. *Reinforcement learning: An introduction*. MIT press.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S. E.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. *CVPR* 1–9.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; and Le, Q. 2018. Mnasnet: Platform-aware neural architecture search for mobile. Technical report, arXiv.

Xiao, L. 2010. Dual averaging methods for regularized stochastic learning and online optimization. *JMLR* 11(Oct):2543–2596.

Xie, L., and Yuille, A. 2017. Genetic CNN. In *ICCV*.

Xie, S.; Zheng, H.; Liu, C.; and Lin, L. 2019. SNAS: stochastic neural architecture search. In *ICLR*.

Yang, Z.; Dai, Z.; Salakhutdinov, R.; and Cohen, W. 2018. Breaking the softmax bottleneck: A high-rank rnn language model. In *ICLR*.

Yao, Q.; Kwok, J.; Gao, F.; Chen, W.; and Liu, T.-Y. 2017. Efficient inexact proximal gradient algorithm for nonconvex problems. In *IJCAI*, 3308–3314. AAAI Press.

Yao, Q.; Wang, M.; Chen, Y.; Dai, W.; Hu, Y.; Li, Y.; Tu, W.-W.; Yang, Q.; and Yu, Y. 2018. Taking human out of learning applications: A survey on automated machine learning. Technical report, arXiv preprint.

Zhong, Z.; Yan, J.; Wu, W.; Shao, J.; and Liu, C.-L. 2018. Practical block-wise neural network architecture generation. In *CVPR*.

Zhou, H.; Yang, M.; Wang, J.; and Pan, W. 2019. BayesNAS: A bayesian approach for neural architecture search. In *ICML*, 7603–7613.

Zoph, B., and Le, Q. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. 2017. Learning transferable architectures for scalable image recognition. In *CVPR*.

A Proofs

A.1 Proposition 1

Proof. Recall that $\mathcal{C} = \mathcal{C}_1 \cap \mathcal{C}_2$ where $\mathcal{C}_1 = \{\mathbf{a} \,|\, \|\mathbf{a}\|_0 = 1\}$ and $\mathcal{C}_2 = \{\mathbf{a} \,|\, 0 \le a_k \le 1\}$, and the proximal step is given by

$$\text{prox}_{\mathcal{C}}(\mathbf{a}) = \mathbf{b}^* = \arg\min_{\mathbf{b}} \frac{1}{2} \|\mathbf{a} - \mathbf{b}\|_2^2 \,, \qquad (12)$$
$$\text{s.t. } \mathbf{a} \in \mathcal{C}_1 \cap \mathcal{C}_2 .$$

Constrain $\mathcal{C}_1$ is means $\mathbf{b}^*$ can be represented as $c\,\mathbf{e}_i$ where c is a parameter to be determined and $\mathbf{e}_i$ is a one-hot vector with the ith element being 1 and all others are zeros; moreover, constrain $\mathcal{C}_2$ means c must be in range $[0, 1]$. Let $\mathbf{a}$ be a k-dimensional vector, then (12) can be decomposed as k separable problems, i.e.,

$$c_i^* = \arg\min_c \frac{1}{2} \|\mathbf{a} - c\,\mathbf{e}_i\|_2^2 \,, \text{ for } i = 1, \dots, k. \qquad (13)$$

where

$$c_i^* = \begin{cases} 0 & \text{if } a_i < 0 \\ a_i & \text{if } 0 \le a_i \le 1 \\ 1 & \text{otherwise} \end{cases} .$$

Let $v_i = \frac{1}{2}(a_i - c_i^*)^2$ and $\mathbf{v} = [v_1, \dots, v_k]$. These contain all k possible solutions for (12). Thus, in order to achieve the minimum, we need to pick up $i^* = \arg\max_i v_i$, and $\mathbf{b}^* = c_{i^*}^* \mathbf{e}_{i^*}$. More compactly, we can express it as

$$\text{prox}_{\mathcal{C}}(\mathbf{a}) = \text{prox}_{\mathcal{C}_1}(\text{prox}_{\mathcal{C}_2}(\mathbf{a})).$$

□

A.2 Theorem 2

Proof. Note that by the definition of loss functions,

- $\mathcal{F}$ is the loss on the validation set, thus $\mathcal{F}$ is bounded from below;
- $\mathcal{F}$ is continuous on $\mathbf{A}$.

Since $\mathbf{A}_t \in \mathcal{C}_2$ and $\max \mathcal{F}(w_t, \mathbf{A}_t) < \infty$, thus $\{\mathbf{A}_t\}$ is constrained within a compact sublevel set. Finally, the Theorem comes from the fact that any infinite sequences on a compact sub-level set must have limit points. □

B Experiment Details

B.1 Datasets

CIFAR-10 CIFAR-10 (Krizhevsky 2009)[1] is a basic dataset for image classification, which consists of 50,000 training images and 10,000 testing images. Half of the CIFAR-10 training images will be utilized as the validation set. Data augmentation like cutout (Devries and Taylor 2017) and HorizontalFlip will be utilized in our experiments. After training, we will test the model on test dataset and report accuracy in our experiments.

PTB PTB[2] is an English corpus used for probabilistic language modeling, which consists of approximately 7 million words of part-of-speech tagged text, 3 million words of skeletally parsed text, over 2 million words of text parsed for predicate-argument structure, and 1.6 million words of transcribed spoken text annotated for speech dis-fluencies. We will choose the model with the best performance on validation dataset and test it on test dataset.

WT2 Compared to the preprocessed version of Penn Treebank (PTB), [3]WikiText-2 (WT2) is over 2 times larger. WT2 features a far larger vocabulary and retains the original case, punctuation and numbers - all of which are removed in PTB.As it is composed of full articles, the dataset is well suited for models that can take advantage of long term dependencies. We will choose the model with the best performance on validation dataset and test it on test dataset.

B.2 Training details

For training CIFAR-10, the convolutional cell consists of N=7 nodes, and the network is obtained by stacking cells for 8 times; in the search process, we train a small network stacked by 8 cells with 50 epochs. SGD is utilized to optimize the network's weights, and Adam is utilized for the parameters of network architecture. To evaluate the performance of searched cells, the searched cells are stacked for 20 times; the network will be fine-tuned for 600 epochs with batch size 96. Additional enhancements like path dropout (of probability 0.2) and auxiliary towers (with weight 0.4) are also used. We have run our experiments for three times and report the mean.

B.3 Search Space

NASP's search space: identity, 1x3 then 3x1 convolution, 3x3 dilated convolution, 3x3 average pooling, 3x3 max pooling, 5x5 max pooling, 7x7 max pooling, 1x1 convolution, 3x3 convolution, 3x3 depthwise-separable conv, 5x5 depthwise-seperable conv, 7x7 depthwise-separable conv.

C More Experiments

C.1 Transferring to Tiny ImageNet

Dataset Tiny ImageNet (Le and Yang 2015)[4] contains a training set of 100,000 images, a testing set of 10,000 images. These images are sourced from 200 different classes of objects from ImageNet. Note that due to small number of training images for each class and low-resolution for images, Tiny ImageNet is harder to be trained than the original ImageNet. Data augmentation like RandomRotation and RandomHorizontalFlip are utilized. After training, we will test the model on test dataset and report accuracy in our experiments.

Results The architecture transferability is important for cells to transfer to other datasets (Zoph et al. 2017). To explore the transferability of our searched cells, we stack

[1] http://www.cs.toronto.edu/~kriz/cifar.html

[2] http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz

[3] https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-2-v1.zip

[4] http://tiny-imagenet.herokuapp.com/

Table 9: Classification accuracy of NASP and state-of-the-art image classifiers on Tiny ImageNet.

Method	Test Accuracy (%)		Params	Search Cost
	top1	top5	(M)	(GPU days)
ResNet18 (He et al. 2016)	52.67	76.77	11.7	-
NASNet-A (Zoph et al. 2017)	**58.99**	**77.85**	4.8	1800
AmoebaNet-A (Real et al. 2018)	57.16	77.62	4.2	3150
ENAS (Pham et al. 2018)	57.81	77.28	4.6	0.5
DARTS (Liu, Simonyan, and Yang 2019)	57.42	76.83	3.9	4
SNAS (Xie et al. 2019)	57.81	76.93	3.3	1.5
NASP	58.12	77.62	4.0	**0.1**
NASP (more ops)	58.32	77.54	8.9	0.2

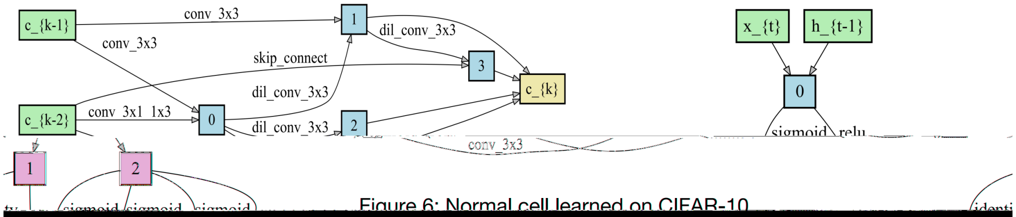


Figure 6: Normal cell learned on CIFAR-10

searched cells for 14 times on Tiny ImageNet, and train the network for 250 epochs. Results are in Tab. 9. We can see NASP exhibits good transferablity, and its performance is also better than other methods except NASNet-A. But our NASP is much faster than NASNet-A.

C.2 Searched Architectures

Architectures identified on CIFAR-10 are shown in Fig.6 and 7, on PTB is shown in Fig.8.