KNAS: Green Neural Architecture Search

Jingjing Xu [1 *] **Liang Zhao** [2] **Junyang Lin** [3] **Rundong Gao** [1] **Xu Sun** [1 2] **Hongxia Yang** [3]

Abstract

Many existing neural architecture search (NAS) solutions rely on downstream training for architecture evaluation, which takes enormous computations. Considering that these computations bring a large carbon footprint, this paper aims to explore a green (namely environmental-friendly) NAS solution that evaluates architectures without training. Intuitively, gradients, induced by the architecture itself, directly decide the convergence and generalization results. It motivates us to propose the gradient kernel hypothesis: Gradients can be used as a coarse-grained proxy of downstream training to evaluate random-initialized networks. To support the hypothesis, we conduct a theoretical analysis and find a practical gradient kernel that has good correlations with training loss and validation performance. According to this hypothesis, we propose a new kernel based architecture search approach KNAS. Experiments show that KNAS achieves competitive results with orders of magnitude faster than "train-then-test" paradigms on image classification tasks. Furthermore, the extremely low search cost enables its wide applications. The searched network also outperforms strong baseline RoBERTA-large on two text classification tasks. Codes are available at https://github.com/Jingjing-NLP/KNAS.

1. Introduction

Neural architecture search (NAS) is a field to automatically explore the optimal architecture (Zoph & Le, 2017; Baker et al., 2017). The search procedure can be divided into three components: search space, optimization approaches, architecture evaluation. Search space defines all network candidates that can be examined to produce the final architecture. The optimization method dictates how to explore the search space. Architecture evaluation is responsible for helping the optimization method to evaluate the quality of architectures. Recent NAS approaches have been able to generate state-of-the-art models on downstream tasks (Zoph et al., 2018; Tan & Le, 2019).

However, despite good performance, NAS usually requires huge computations to find the optimal architecture, most of which are used for architecture evaluation. For example, Zoph & Le (2017) use 800 GPUs for 28 days resulting in 22,400 GPU-hours. Strubell et al. (2019) find that a single neural architecture search solution can emit as much carbon as five cars in their lifetimes. The major recent advance of NAS is to reduce the search costs. One research line focuses on search methods without architecture evaluation, like DART (Liu et al., 2019; Chu et al., 2020c). While being simple, previous studies have shown these approaches suffer from well-known performance collapse due to an inevitable aggregation of skip connections (Ying et al., 2019; Chu et al., 2020b). Another line focuses on reducing architecture evaluation costs by using techniques like early-stopping (Liang et al., 2019), weight-sharing (Pham et al., 2018), performance predicting (Liu et al., 2018), and so on. Compared to the original full-training solution, these techniques achieve promising speedup. However, if we consider a complicated search space with plenty of networks, they still require many computations.

In this work, we aim to explore a more challenging question: Can we evaluate architectures without training? Before answering this question, it is essential to figure out how architecture affects optimization. It is widely accepted that gradients, induced by neural networks, play a crucial role in optimization (Bengio et al., 1994; Hochreiter & Schmidhuber, 1997; Pascanu et al., 2013). Motivated by this common belief, we propose a bold hypothesis: gradients can be used as a coarse-grained proxy of downstream training to evaluate randomly-initialized architectures. To support the hypothesis, we conduct a comprehensive theoretical analysis and identify a key gradient feature, the Gram matrix of gradients (GM). For any neural networks, the F-norm of GM decides the upper bound of convergence rate. Higher F-norm is expected for a higher convergence rate. Intuitively, GM can be regarded as a health index of gradients. Each element in GM is the dot product between any two gradient

[*]The joint work between Peking University and Alibaba Group. [1]MOE Key Lab of Computational Linguistics, School of EECS, Peking University [2]Center for Data Science, Peking University [3]Alibaba Group. Correspondence to: Jingjing Xu <jingjingxu@pku.edu.cn>, Xu Sun <xusun@pku.edu.cn>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

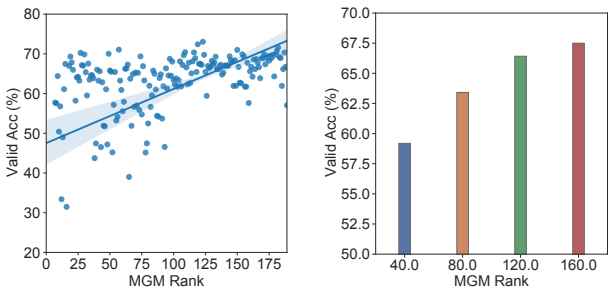


Figure 1. An illustration of the relation between MGM and accuracy. We sample 200 architectures and rank them based on MGM scores. Smaller rank represents smaller MGM. Y-axis lists the validation accuracy. The right figure classifies architectures into four groups based on their MGM ranks. Each bar represents the average accuracy. The Spearman coefficient is 0.56 ($P \ll 0.01$), indicating good positive correlations.

vectors. Vanishing gradients and ataxic gradients both get lower GM scores. Following the theoretical results, we propose to leverage GM to evaluate randomly-initialized networks. To be specific, the mean of GM, short for MGM, is adopted in implementation. We conduct experiments on CIFAR100 and calculate the Spearman correlation scores between MGM and key optimization metrics. Experiments show that MGM has good correlations with negative training loss and validation accuracy, with 0.53 ($P \ll 0.01$) and 0.56 ($P \ll 0.01$) coefficients, respectively[1]. Figure 1 illustrates the relation between MGM and validation accuracy. These results are strong evidence to support the hypothesis, demonstrating that MGM has good potential to be used as a coarse-grained architecture evaluation.

According to our hypothesis, we propose a gradient kernel based NAS solution, short for KNAS. We first select top-k architectures with the largest MGM as candidates, which are then trained to choose the best one. In practice, k is usually set to be a very small value. Experiments show that the new approach achieves large speedups with competitive accuracies on NAS-Bench-201 (Dong & Yang, 2020), a NAS benchmark dataset. Furthermore, the low search cost allows us to apply KNAS to diverse tasks. We find that the structures searched by KNAS outperform strong baseline RoBERTA-large on two text classification tasks.

The main contributions are summarized as follows:

- We propose a gradient kernel hypothesis to explore training-free architecture evaluation approaches.
- We find a practical gradient feature MGM to support the hypothesis.
- Based on the hypothesis, we propose a green NAS solution KNAS that can find well-performing architectures with orders of magnitude faster than "train-then-test" NAS solutions.

[1] https://www.statstutor.ac.uk/resources/uploaded/spearmans.pdf

2. Related Work

Network architecture search Neural architecture search aims to replace expert-designed networks with learned architectures (Chang et al., 2019; Li et al., 2019; Zhou et al., 2020; He et al., 2020a; Fang et al., 2020; He et al., 2020b; You et al., 2020; Alves & de Oliveira, 2020). The first NAS research line mainly focuses on random search. The key idea is to randomly evaluate various architectures and select the best one based on their validation results. However, despite promising results, many of them require thousands of GPU days to achieve desired results. To address this problem, Zoph & Le (2017) propose a reinforcement learning based search policy that introduces an architecture generator with validation accuracy as a reward.

Another research line is based on evolution approaches. Real et al. (2019) propose a two-stage search policy. The first stage selects several well-performing parent architectures. The second stage applies mutation on these parent architectures to select the best one. Following this work, So et al. (2019) apply the evolution search on Transformer networks and achieve new state-of-the-art results on machine translation and language modeling tasks. Although these approaches can reduce exploration costs, the dependence on downstream training still leads to huge computation costs.

To fully get rid of the dependence on validation accuracy, several studies (Jiang et al., 2019; Liu et al., 2019; Dong & Yang, 2019; Zela et al., 2020; Chu et al., 2020a) re-formulate the task in a differentiable manner and allow efficient search using gradient descent. Furthermore, Unlike these studies, we propose a lightweight NAS approach, which largely reduces evaluation costs.

There are three corresponding studies similar to our approach (Mellor et al., 2020; Chen et al., 2021; Abdelfattah et al., 2021). Mellor et al. (2020) evaluate randomly-initialized architectures based on the output of rectified linear units. In contrast, KNAS uses gradient kernels to evaluate architectures. Moreover, KNAS does not require any architecture constraints. Chen et al. (2021) propose to rank randomly-initialized architectures by analyzing the spectrum of the neural tangent kernel and the number of linear regions in the input space. Different from this approach, KNAS uses the Gram matrix of gradients to rank architectures. Abdelfattah et al. (2021) evaluates several conventional reduced-training proxies for ranking randomly-initialized models.

Understanding and improving optimization Understanding and improving the optimization of deep networks has long been a hot research topic. Bengio et al. (1994) find the vanishing and exploding gradient problem in neural network training. A lot of advanced optimization approaches have been proposed in recent years, which can be classified into four research lines. The first research line focuses on initialization (Sutskever et al., 2013; Mishkin & Matas, 2016; Hanin & Rolnick, 2018). Glorot & Bengio (2010) propose to control the variance of parameters via appropriate initialization. Following this work, several widely-used initialization approaches have been proposed, including Kaiming initialization (He et al., 2015), Xaiver initinalization (Glorot & Bengio, 2010), and Fixup initialization (Zhang et al., 2019). The second research line focuses on normalization (Ioffe & Szegedy, 2015; Lei Ba et al., 2016; Ulyanov et al., 2016; Wu & He, 2018; Nguyen & Salazar, 2019). These approaches aim to avoid the vanishing gradient by controlling the distribution of intermediate layers. The third is mainly based on activation functions to make the derivatives of activation less saturated to avoid the vanishing problem, including GELU activation (Hendrycks & Gimpel, 2016), SELU activation (Klambauer et al., 2017), and so on. The motivation behind these approaches is to avoid unsteady derivatives of the activation concerning the inputs. The fourth focuses on gradient clipping (Pascanu et al., 2013).

Gradient Kernel Our work is also related to gradient kernels (Advani & Saxe, 2017; Jacot et al., 2018). NTK (Jacot et al., 2018; Du et al., 2019b) is a popular gradient kernel, defined as the Gram matrix of gradients. It is proposed to analyze the model's convergence and generalization. Following these studies, many researchers are devoted to understand current networks from the perspective of NTK (Lee et al., 2019; Hastie et al., 2019; Allen-Zhu et al., 2019; Arora et al., 2019). Du et al. (2019b) use the Gram matrix of gradients to prove that for an m hidden node shallow neural network with ReLU activation, as long as m is large enough, randomly initialized gradient descent converges to a globally optimal solution at a linear convergence rate for the quadratic loss function. Following this work, Du et al. (2019a) further expand this finding and prove that gradient descent achieves zero training loss in polynomial time for a deep over-parameterized neural network with residual connections.

3. Gradient Kernel Hypothesis

In this section, we introduce a gradient kernel hypothesis for training-free architecture evaluation. It is widely believed that gradients, induced by neural networks, are the most direct factor to decide convergence and generalization results (Yuan et al., 2016). It motivates us to explore whether gradients can be used as an alternative replacement of downstream training to evaluate architectures.

We consider a simple multi-layer fully-connected neural network with L layers. Assume that the output of h-th layer is:

$$\boldsymbol{y}^{(h)} = \sigma(\boldsymbol{w}^{(h)}\boldsymbol{y}^{(h-1)}), \tag{1}$$

where σ is the activation function and $\boldsymbol{w}^{(h)}$ is the weight matrix. The gradient for the h-th layer with respect to the weight matrix $\boldsymbol{w}^{(h)}$ is:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^{(h)}} &= \langle \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(L)}}, \boldsymbol{y}^{(h-1)} \frac{\partial \boldsymbol{y}^{(L)}}{\partial \boldsymbol{y}^{(h)}} \boldsymbol{J}^{(h)} \rangle \\ &= \langle \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(L)}}, \boldsymbol{y}^{(h-1)} (\prod_{k=h+1}^{L} \boldsymbol{J}^{(k)} \boldsymbol{w}^{(k)}) \boldsymbol{J}^{(h)} \rangle,\end{aligned} \tag{2}$$

where $\mathcal{L}$ is the loss function and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(L)}}$ is the derivative of the output of the last layer[2]. $\boldsymbol{y}^{(h-1)}$ is the output of $h-1$ layer. $\boldsymbol{J}^{(k)}$ is the diagonal matrix where the main diagonal is the list of the derivatives of the activation with respect to the inputs in the k-th layer:

$$\boldsymbol{J}^{(k)} = \mathbf{diag}(\sigma'(\boldsymbol{w}_1^{(k)}\boldsymbol{y}^{(k-1)}), \cdots, \sigma'(\boldsymbol{w}_d^{(k)}\boldsymbol{y}^{(k-1)})), \tag{3}$$

where d is the dimension of the activation and $\boldsymbol{w}_d^{(k)}$ is the d-th row in matrix $\boldsymbol{w}^{(k)}$ in the k layer.

From this equation, we can see that gradients depend on the design of networks, including network depth, activation function, initialization, and so on. Therefore, different architectures will result in totally different gradients. Motivated by this, we propose a gradient kernel hypothesis:

Proposition 3.1 (Gradient Kernel Hypothesis)**.** *Suppose $\mathcal{G}$ is a set of all gradient features. There exists a gradient feature $g \in \mathcal{G}$ that can be used as a coarse-grained proxy of downstream training to evaluate randomly-initialized architectures.*

4. Identify Key Gradient Kernels

To support the hypothesis, we follow the idea of NTK (Jacot et al., 2018) and show an theoretical understanding about the relationship between architectures and convergence results. Theoretical results show that the Gram matrix of gradients, short for GM, decides the convergence results. It is a good signal showing that GM is likely to be a good proxy of downstream performance to evaluate the quality of architectures.

Formally, we consider a neural network with L layers. On the h-th layer, the output of $\boldsymbol{y}^{(h+1)}$ is computed as:

$$\boldsymbol{y}^{(h+1)} = \sigma(\boldsymbol{w}^{(h)}\boldsymbol{y}^{(h)}), \tag{4}$$

[2]Here we adopt numerator layout notation.

where $\boldsymbol{y}_0 = \boldsymbol{x}$ and $\boldsymbol{x} \in \mathcal{R}^d$. $\boldsymbol{w}^{(h)} \in \mathcal{R}^{d\times d}$ is a trainable parameter matrix. The last layer is a sum layer responsible for generating the final label $y^{(L)}$. We focus on the empirical risk minimization problem with a quadratic loss. Given a training dataset $(\boldsymbol{x}, \boldsymbol{y})_{i=1}^n$, we train the model by minimizing MSE loss:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{y}^{(L)} - \boldsymbol{y}^*\|_2^2, \tag{5}$$

where $\boldsymbol{y}^* = \{y_1^*, \cdots, y_n^*\}$ is the gold label vector. $\boldsymbol{y} = \{y_1^{(L)}, \cdots, y_n^{(L)}\}$ is the prediction label vector. n is the size of training data. We apply gradient descent to optimize weights:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \mu\frac{\partial\mathcal{L}(\boldsymbol{w}(t))}{\partial\boldsymbol{w}(t)}, \tag{6}$$

where t represent the t-th iteration, $\mu > 0$ is the step size, $\boldsymbol{w}(t)$ is the parameter vector at the t-th iteration. The gradient vector is:

$$\frac{\partial\mathcal{L}(\boldsymbol{w}(t), i)}{\partial(\boldsymbol{w}(t))} = (y_i^{(L)} - y_i^*)\frac{\partial y_i^{(L)}}{\partial\boldsymbol{w}(t)}, \tag{7}$$

where $\mathcal{L}(\boldsymbol{w}(t), i)$ is the loss function of example i. We consider gradient descent with infinitesimal step size following Arora et al. (2018). Formally, we consider the ordinary differential equation defined by:

$$\frac{d\boldsymbol{w}(t)}{dt} = -\frac{\partial\mathcal{L}(\boldsymbol{w}(t), i)}{\partial\boldsymbol{w}(t)}. \tag{8}$$

Define a matrix $\boldsymbol{H}$ where the entry (i, j) is:

$$\boldsymbol{H}_{i,j}(t) = \left(\frac{\partial y_j^{(L)}(t)}{\partial\boldsymbol{w}(t)}\right)\left(\frac{\partial y_i^{(L)}(t)}{\partial\boldsymbol{w}(t)}\right)^T. \tag{9}$$

Suppose $\boldsymbol{g}_i = \frac{\partial y_i^{(L)}}{\partial\boldsymbol{w}(t)}$ and $\boldsymbol{g}_j = \frac{\partial y_j^{(L)}}{\partial\boldsymbol{w}(t)}$. $\boldsymbol{H}_{i,j}(t)$ is the dot-product between two vectors $\boldsymbol{g}_i$ and $\boldsymbol{g}_j$. Thus, $\boldsymbol{H}$ can be regarded as a Gram matrix.

Our first step is to calculate the dynamics of each prediction:

$$\begin{aligned}\frac{d}{dt}y_j^{(L)}(t) &= \langle\frac{\partial y_j^{(L)}(t)}{\partial\boldsymbol{w}(t)}, \frac{d\boldsymbol{w}(t)}{dt}\rangle \\ &= \left(\frac{\partial y_j^{(L)}(t)}{\partial\boldsymbol{w}(t)}\right)\left(\frac{\partial y_i^{(L)}(t)}{\partial\boldsymbol{w}(t)}\right)^T(y_i^* - y_i^{(L)}),\end{aligned} \tag{10}$$

Given $\boldsymbol{H}$, Eq. 10 can be written as:

$$\frac{d}{dt}y_j^{(L)}(t) = \boldsymbol{H}_{i,j}(t)(y_i^* - y_i^{(L)}). \tag{11}$$

$\boldsymbol{H}_{i,j}(t)$ is the dot-product between two vectors $\boldsymbol{g}_i$ and $\boldsymbol{g}_j$. We write the dynamic of predictions in a compact way:

$$\frac{d}{dt}\boldsymbol{y}^{(L)}(t) = \boldsymbol{H}(t)(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)). \tag{12}$$

According to matrix $\boldsymbol{H}$ and Eq. 12, we have the following proposition.

Proposition 4.1. *Suppose $\boldsymbol{H}(t)$ is the Gram matrix of gradients. $\forall t > 0$, the following inequation holds:*

$$\|(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))\|_2^2 \leq exp(-\lambda_{min}(\boldsymbol{H}(t))t)\|(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(0))\|_2^2. \tag{13}$$

Refer to Appendix D for detailed proofs. We can see that the upper bound of losses is decided by $\lambda_{min}(\boldsymbol{H}(t))$ and larger $\lambda_{min}(\boldsymbol{H}(t))$ is expected for lower training losses. Furthermore, since $\boldsymbol{H}(t)$ is symmetrical, the F-norm of $\boldsymbol{H}(t)$ bounds $\lambda_{min}(\boldsymbol{H}(t))$ by

$$\lambda_{min}(\boldsymbol{H}(t)) \leq \sqrt{\sum_i |\lambda_i|^2} = \|\boldsymbol{H}(t)\|_F, \tag{14}$$

where λ_i is the eigenvalue of $\boldsymbol{H}(t)$.

As we can see, the F-norm of GM bounds the convergence rate. Each entry in $\boldsymbol{H}(t)$ is the dot-product between two gradient vectors, which is decided by two parts, gradient values and gradient correlations. Intuitively, GM is the health metric of gradients. From the viewpoint of geometric, gradient values decide the step size in optimization, and gradient correlations evaluate gradient directions' randomness. Extremely small gradient values will stop training and decrease the convergence rate. Extremely small gradient correlations mean "random" gradient directions, which bring conflicted parameter updates and increase the training cost. Higher F-Norm values are expected for a higher convergence rate. To conclude, the theoretical results demonstrate that the gram matrix of gradients is an crucial and comprehensive measurement to evaluate the quality of architectures.

5. KNAS: Support the Hypothesis

To support our hypothesis, we propose to leverage GM to evaluate randomly-initialized networks. In implementation, we use the mean of GM, short for MGM, as the final feature and calculate the correlation score between MGM and key optimization features.

Definition 5.1 (Gradient Kernel)**.** *Suppose $\boldsymbol{H}$ is the Gram matrix of gradients where each item (i, j) represents the dot-product of two gradient vectors. Gradient kernel $\boldsymbol{g}$ is the mean of all elements in $\boldsymbol{H}$:*

$$\boldsymbol{g} = \frac{1}{n^2}\sum_{i=1}^n\sum_{j=1}^n\left(\frac{\partial y_j^{(L)}(t)}{\partial\boldsymbol{w}(t)}\right)\left(\frac{\partial y_i^{(L)}(t)}{\partial\boldsymbol{w}(t)}\right)^T. \tag{15}$$

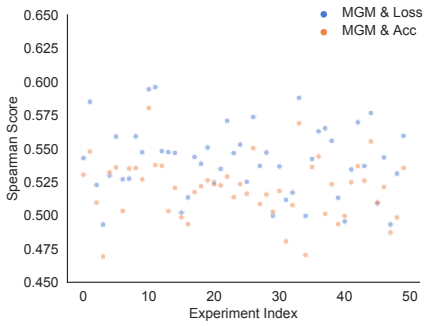


Figure 2. The relation between MGM and key optimization results, including training loss and validation accuracy. "Loss" represents negative training loss, and "Acc" represents validation accuracy. We conduct 50 experiments, each sampling 200 architectures. As we can see, MGM has good positive correlations with training loss and validation accuracy, with around 0.53 and 0.55 Spearman coefficients ($p \ll 0.01$).

In implementation, since the length of the whole gradient vectors is too long, we approximate Eq. 15 as:

$$\boldsymbol{g} = \frac{1}{Mn^2} \sum_{m=1}^{M} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\frac{\partial y_j^{(L)}(t)}{\partial \hat{\boldsymbol{w}}^m(t)} \right) \left(\frac{\partial y_i^{(L)}(t)}{\partial \hat{\boldsymbol{w}}^m(t)} \right)^T . \tag{16}$$

where K is the number of layers. We take each layer as a basic unit and compute the mean of the Gram matrix for each layer. $\hat{\boldsymbol{w}}^m$ is the sampled parameters from the m-th layer where the length of $\hat{\boldsymbol{w}}^m$ is set to 50 in implementation.

To support the hypothesis, we sample 200 architectures from a NAS benchmark, NAS-Bench-201, and show the relation between kernel values and optimization results on CIFAR100. Figure 2 shows that MGM has good correlations with training loss and validation accuracy, which supports our hypothesis.

We then propose a lightweight kernel-based NAS solution, called KNAS. The details are shown in Algorithm 1. First, we generate s possible architectures as the search space $\mathcal{S}$. We do not fix the generating policy to keep the flexibility on diverse tasks and scenarios in this work. For each architecture, we calculate MGM based on Eq. 16 and select k architectures with the highest scores as candidates, which are then trained from scratch to get their results on a validation set. MGM evaluation does not need any training steps. Compared to other NAS approaches that train hundreds of architectures, the new approach can largely reduce search costs.

6. Experiments

In this section, we evaluate the proposed approach on NAS-Bench-201 (Dong & Yang, 2020), which provides a fair setting for evaluating different NAS techniques.

Algorithm 1 KNAS Algorithm

Require: Search space $\mathcal{S}$, training set $\mathcal{D}_t$, validation set $\mathcal{D}_v$
Initialize: max_iteration = M
Initialize candidate set $\mathcal{C}$ = []
for search_iteration in 1, 2, ..., max_iteration **do**
  Randomly sample an architecture s from $\mathcal{S}$
  Compute MGM based on Eq. 15
  $\mathcal{C}$.append(s, MGM)
  update $\mathcal{C}$ to ensure only top-k architectures are kept
end for
A^* = top_1($\mathcal{C}$,$\mathcal{D}_t$,$\mathcal{D}_v$) # Choose the best architecture based on validation performance.
return A^*

6.1. Datasets

NAS-Bench-201 (Dong & Yang, 2020) is a benchmark dataset for NAS algorithms, constructed on image classification tasks, including CIFAR10, CIFAR100, and ImageNet-16-120 (ImageNet-16). CIFAR10 and CIFAR100 are two widely used datasets[3]. CIFAR-10 consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. CIFAR100 has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. ImageNet-16 is provided by Chrabaszcz et al. (2017). NAS-Bench-201 chooses four nodes and five representative operation candidates for the operation set, which generates 15,625 cells/architectures as search space. Each architecture contains full training logs, validation accuracy, and test accuracy on CIFAR10, CIFAR100, and ImageNet-16. In summary, NAS-Bench-201 enables researchers to easily re-implement previous approaches via providing all architecture evaluation results. However, since these results are unavailable in real-world tasks, we take the evaluation time into account when computing the search time for a fair comparison.

6.2. Baselines

We compare the proposed approach with the following baselines:

Random Search Algorithm It includes two baselines, random search (RS) (Bergstra & Bengio, 2012) and random search with parameter sharing (RSPS) (Li & Talwalkar, 2019). Random search is a simple baseline that randomly samples several architectures and selects the network with the best validation accuracy as the final architecture. RSPS

[3]https://www.cs.toronto.edu/ kriz/cifar.html

Table 1. Results on CIFAR10, CIFAR100, and ImageNet-16. "Time" means the total search time. The proposed approach KNAS achieves competitive results with the lowest costs. All search steps in KNAS can be finished within a few hours. Accuracies of NASWOT and TE-NAS come from their original papers.

Type	Model	CIFAR10			CIFAR100			ImageNet-16		
		Acc	Time(s)	Speed-up	Acc	Time(s)	Speed-up	Acc	Time(s)	Speed-up
w/o Search	ResNet	**93.97**	N/A	N/A	70.86	N/A	N/A	43.63	N/A	N/A
Search	RS	93.63	216K	1.0x	71.28	460K	1.0x	44.88	1M	1.0x
	RL	92.83	216K	1.0x	70.71	460K	1.0x	44.10	1M	1.0x
	REA	93.72	216K	1.0x	**72.12**	460K	1.0x	45.01	1M	1.0x
	BOHB	93.49	216K	1.0x	70.84	460K	1.0x	44.33	1M	1.0x
	RSPS	91.67	10K	21.6x	57.99	46K	21.6x	36.87	104K	9.6x
Gradient	GDAS	93.36	22K	12.0x	67.60	39K	11.7x	37.97	130K	7.7x
	DARTS	88.32	23K	9.4x	67.34	80K	5.8x	33.04	110K	9.1x
Training-free	NASWOT	92.96	2.2K	100x	70.03	4.6K	100x	44.43	10K	100x
	TE-NAS	93.90	2.2K	100x	71.24	4.6K	100x	42.38	10K	100x
MGM	**KNAS** ($k = 2$)	93.05	4.2K	50x	68.91	9.2K	50x	34.11	20K	50x
	KNAS ($k = 5$)	93.42	10.8K	20x	71.42	23K	20x	**45.35**	50K	20x

introduces a two-stage training process. The first is a shared-parameter training stage. The second is a search stage, which chunks pre-trained modules and evaluates all possible architectures on a validation set. The architecture with the highest validation accuracy is selected as the final architecture.

Reinforcement learning based algorithm It includes one baseline, RL (Williams, 1992). RL is a method that introduces an architecture generator to generate well-performing architectures. To train the generator, it uses validation accuracy as reward.

Evolution-based search algorithm It includes one baseline: regularized evolution for architecture search (REA) (Real et al., 2019). It first selects and trains several parent architectures and then applies mutation operations on these parent architectures to get child architectures. The architecture with the best validation accuracy is adopted as the final architecture.

Differentiable algorithm It includes two baselines, DARTS (Liu et al., 2019) and GDAS (Dong & Yang, 2019). DARTS reformulates the search problem into a continues search problem. It uses one-batch gradients to replace the accuracy on a validation set to guide a model how to search for well-performing architectures. Due to unsteady training, the results provided by the original NAS-Bench-201 paper are very low. To avoid model collapse, we use different seeds to train the whole framework multiple times and generate multiple candidate architectures, which are then trained from scratch to select the best one. Following this work, GDAS updates a sub-graph rather than the whole graph for higher speedups.

Hyper-parameter optimization algorithm It includes one baseline, BOHB (Falkner et al., 2018). This approach combines the benefits of Bayesian optimization and bandit-based methods. It requires "train-then-test" paradigm to evaluate different architectures. We use the code provided by Dong & Yang (2020) for baseline implementation.

Training-free algorithm It includes two approaches, NASWOT and TE-NAS. NASWOT uses the output of rectified linear units to evaluate architectures. TE-NAS uses the spectrum of the neural tangent kernel and the number of linear regions in the input space to evaluate architectures.

6.3. Experiment Settings

We randomly sample 100 architectures as the search space in the proposed approach and calculate their MGM scores at initialization. All baselines are implemented on a single NVIDIA V100 GPU. We use the released code provided by Dong & Yang (2020) for baseline implementation.

Architecture evaluation contains two steps: network training and evaluating. For all approaches, we set the time of architecture training plus evaluation to 2,160 seconds, 4,600 seconds, and 10,000 seconds on CIFAR10, CIFAR100, and ImageNet-16, respectively. Here we do not adopt the reported time in Dong & Yang (2020) for architecture evaluation because the authors use multiple workers to train a model. In contrast, we adopt a single GPU (Tesla-V100) and a single worker. We observe that architecture evaluation takes almost all search time in search-based approaches (including RS, RL, REA, BOHB, RSPS). The rest operations only take a few seconds, which are not included in the search time for simplicity. For DARTS and RSPS, we slightly update the search policy for better performance. We

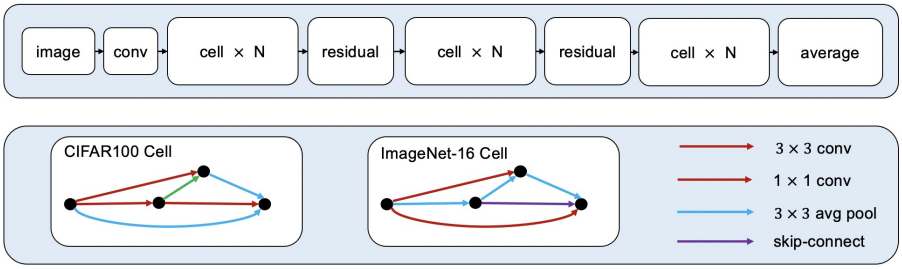
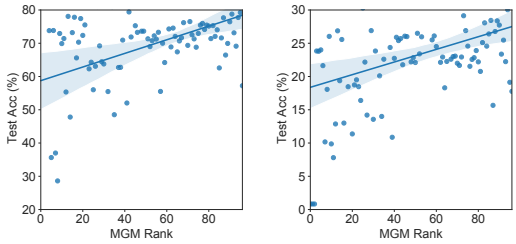


Figure 3. The searched architectures for CIFAR100 and ImageNet-16. They share the same architecture framework (shown in the top block). The bottom block shows the details of the searched cell.



(a) CIFAR100 (b) ImageNet-16

Figure 4. MGM has good correlations with downstream performance on CIFAR100 and ImageNet-16. Smaller rank represents smaller MGM. Y-axis lists test accuracy.

run DARTS and RSPS up to 10 epochs and evaluate the selected architecture after each epoch.

7. Results

Table 1 shows the comparison between the proposed approach and strong baselines. RS is a naive baseline, which randomly selects architectures for full-training and evaluation. It is one of the most time-consuming methods and achieves good accuracy improvements with 0.46 and 1.25 gains on CIFAR100 and ImageNet-16. RL adds reinforcement learning into the search process and is able to explore more well-performing architectures. However, due to its unsteady training, RL does not beat RS on all datasets in terms of accuracy. REA contains two search stages: parent search and mutation. The second stage explores new architectures based on the best-performing parent architectures. Roughly speaking, better parent networks have better inductive bias and thus provide student networks with good initialization. This approach achieves the highest performance with 93.72, 72.12, and 45.01 scores on CIFAR10, CIFAR100, and ImageNet-16. However, despite promising results, these baselines require considerable time for architecture evaluation. Besides, several approaches also achieve much better speed-up results, such as RSPS, GDARTS, and DARTS. However, the accuracy of architectures searched by these approaches is the main problem.

Unlike these approaches, the proposed approach achieves competitive results with the lowest costs. Compared to the time-consuming approaches, including RS, RL, REA, and BOHB, KNAS with k=5 brings around 20X speedups on CIFAR10, CIFAR100 ImageNet-16. Compared to approaches with lower search costs, including RSPS, GDAS, DARTS, NASWOT, and TE-NAS, the proposed approach achieves large accuracy improvements. The searched architecture also outperforms ResNet with 0.56 and 1.72 accuracy improvements on CIFAR100 and ImageNet-16. Figure 3 visualizes the architectures searched by KNAS. Furthermore, we also visualize the relation between MGM and downstream performance on CIFAR100 and ImageNet-16. Results are shown in Figure 4. Each dot represents a single architecture with its MGM value (X-axis) and accuracy (Y-axis). As we can see, the proposed hypothesis also holds on IFAR10 and ImageNet-16. They both have good correlations between MGM and downstream performance.

8. Discussion: Generalization on Diverse Tasks

To avoid the bias on specific factors, like initialization, we conduct more experiments on text classification tasks to verify the generalization ability of KNAS. The settings of text classification are different from that of NAS-Bench-201. For example, NAS-Bench-201 adopts Kaiming initialization, CNN cells, batch normalization, while text classification adopts Xavier initialization, self-attention cells, layer normalization. Text classification includes two datasets, MRPC evaluating whether two sentences are semantically equivalent and RTE recognizing textual entailment. Two classification datasets are provided by Wang et al. (2019).

8.1. Datasets and Baselines

Text classification datasets is provided by GLUE (Wang et al., 2019), a platform designed for natural language understanding. We use MRPC and RTE in this work. MRPC

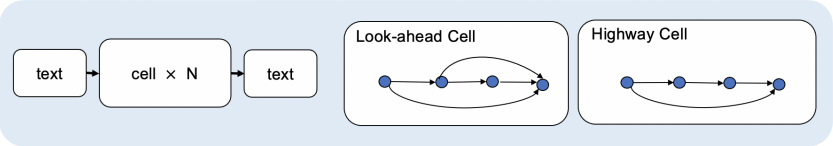


Figure 5. An illustration of look-ahead networks and high-way networks. They have the same architecture framework where all cells are connected in a chain way. For each look-ahead cell, the last layer takes all inner outputs as the input. Highway cells are similar to look-ahead cells except for the last layer only taking the first layer output and the last layer output as the input.

is a classification dataset for evaluating whether two sentences are semantically equivalent. Each example has two sentences and a human-annotated label. We use accuracy as the evaluation metric. RTE is a textual entailment dataset. It comes from a series of annual textual entailment challenges. Following Wang et al. (2019), we combine the data from RTE1, RTE2, RTE3, and RTE5 and adopt the same data-processing steps. We adopt RoBERTA-large, a widely-used pre-trained model, as the baseline. The code is provided by HuggingFace[4], a widely used pre-training model project. All network candidates are initialized with pre-trained parameters. We run Roberta-large on 8 V100 GPUs. For each dataset, we randomly run Roberta-large 5 times and report the average results. For MRPC, the batch size is set to 4, and the learning rate is set to $3e-5$. For RTE, the batch size is set to 4, and the learning rate is set to $2e-5$. For the rest hyper-parameters, we use the default settings.

8.2. Search Space

We define several skip-connection structures, each with possible candidate architectures, as shown in Figure 5. We merge these architectures as a search space. We adopt the same settings for all architectures: 12 encoder layers and 12 decoder layers. These architectures also share the same hype-parameters. Here are the details of different structures.

Highway networks It connects all cells in a chain way. In each cell, there exists a highway connection from the first layer to the last layer. The number of layers in each cell ranges from 2 to 11, and there are 10 possible architectures.

Look-ahead networks In each cell, the layer is connected in a chain way except for the last layer. The last layer takes all outputs of previous layers in a cell as inputs. The number of layers in each cell ranges from 2 to 11, and there are 10 possible architectures.

DenseNet It splits the whole network into different cells (Huang et al., 2017). We adopt the original DenseNet structure and omit its implementation details in Figure 5 for simplification. The input of the current cell comes from all outputs of previous cells. In each cell, the layer is connected in a chain way. The number of layers in each cell ranges from 2 to 11.

[4]https://github.com/huggingface/transformers

Table 2. Results on text classification tasks. "Time" refers to the search time.

Models	MRPC		RTE	
	Acc	Time(s)	Acc	Time(s)
RoBERTA-large	92.08	N/A	83.51	N/A
KNAS	**93.32**	0.4K	**83.75**	2K

8.3. Results

The results are shown in Table 2. The architectures searched by the proposed approach achieve better results on all datasets. Though pre-trained models are widely believed to be state-of-the-art approaches, KNAS still achieves performance improvements over RoBERTA-large with 1.24 and 0.24 accuracy improvements on MRPC and RTE datasets.We choose top-2 and top-5 architectures for full training and evaluation for two classification datasets. It proves that the proposed approach generalizes well on various datasets.

9. Conclusion

In this work, we aim to explore a green NAS solution by getting rid of downstream training from architecture evaluation. We propose a hypothesis that gradients can be used to evaluate randomly-initialized networks. To support the hypothesis, we conduct a theoretical analysis and find an appropriate feature MGM. According to this feature, we develop a simple yet efficient architecture search approach KNAS. It achieves large speedups with competitive accuracies on a NAS benchmark dataset. Furthermore, KNAS generalizes well and can search for better architectures on text classification tasks.

Acknowledgement

We thank the anonymous reviewers for their constructive suggestions and comments. This work is partly supported by Beijing Academy of Artificial Intelligence (BAAI).

References

Abdelfattah, M. S., Mehrotra, A., Dudziak, L., and Lane, N. D. Zero-cost proxies for lightweight NAS. *ICLR*, 2021.

Advani, M. S. and Saxe, A. M. High-dimensional dynamics of generalization error in neural networks. *CoRR*, abs/1710.03667, 2017.

Allen-Zhu, Z., Li, Y., and Liang, Y. Learning and generalization in overparameterized neural networks, going beyond two layers. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 6155–6166, 2019.

Alves, J. H. and de Oliveira, L. F. Optimizing neural architecture search using limited GPU time in a dynamic search space: A gene expression programming approach. *CoRR*, abs/2005.07669, 2020.

Arora, S., Cohen, N., and Hazan, E. On the optimization of deep networks: Implicit acceleration by overparameterization. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 244–253, 2018.

Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 322–332. PMLR, 2019.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=S1c2cvqee.

Bengio, Y., Simard, P. Y., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166, 1994.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

Chang, J., Zhang, X., Guo, Y., Meng, G., Xiang, S., and Pan, C. DATA: differentiable architecture approximation. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 874–884, 2019.

Chen, W., Gong, X., and Wang, Z. Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective. *ICLR*, 2021.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017.

Chu, X., Zhang, B., and Li, X. Noisy differentiable architecture search. *CoRR*, abs/2005.03566, 2020a.

Chu, X., Zhou, T., Zhang, B., and Li, J. Fair DARTS: eliminating unfair advantages in differentiable architecture search. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J. (eds.), *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XV*, volume 12360 of *Lecture Notes in Computer Science*, pp. 465–480. Springer, 2020b.

Chu, X., Zhou, T., Zhang, B., and Li, J. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *European Conference on Computer Vision*, pp. 465–480, 2020c.

Dong, X. and Yang, Y. Searching for a robust neural architecture in four GPU hours. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 1761–1770, 2019.

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Du, S. S., Lee, J. D., Li, H., Wang, L., and Zhai, X. Gradient descent finds global minima of deep neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 1675–1685, 2019a.

Du, S. S., Zhai, X., Póczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019b.

Falkner, S., Klein, A., and Hutter, F. BOHB: robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 1436–1445, 2018.

Fang, J., Sun, Y., Peng, K., Zhang, Q., Li, Y., Liu, W., and Wang, X. Fast neural network adaptation via parameter remapping and architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pp. 249–256, 2010.

Hanin, B. and Rolnick, D. How to start training: The effect of initialization and architecture. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 569–579, 2018.

Hastie, T., Montanari, A., Rosset, S., and Tibshirani, R. J. Surprises in high-dimensional ridgeless least squares interpolation. *CoRR*, abs/1903.08560, 2019.

He, C., Annavaram, M., and Avestimehr, S. Fednas: Federated deep learning via neural architecture search. *CoRR*, abs/2004.08546, 2020a.

He, C., Ye, H., Shen, L., and Zhang, T. Milenas: Efficient neural architecture search via mixed-level reformulation. *CoRR*, abs/2003.12238, 2020b.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1026–1034, 2015.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456, 2015.

Jacot, A., Hongler, C., and Gabriel, F. Neural tangent kernel: Convergence and generalization in neural networks. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 8580–8589, 2018.

Jiang, Y., Hu, C., Xiao, T., Zhang, C., and Zhu, J. Improved differentiable architecture search for language modeling and named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 3583–3588, 2019.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 972–981, 2017.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 8570–8581, 2019.

Lei Ba, J., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Li, G., Qian, G., Delgadillo, I. C., Müller, M., Thabet, A. K., and Ghanem, B. SGAS: sequential greedy architecture search. *CoRR*, abs/1912.00195, 2019.

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, pp. 129, 2019.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., and Li, Z. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A. L., Huang, J., and Murphy, K. Progressive neural architecture search. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y. (eds.), *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I*, volume 11205 of *Lecture Notes in Computer Science*, pp. 19–35, 2018.

Liu, H., Simonyan, K., and Yang, Y. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Mellor, J., Turner, J., Storkey, A. J., and Crowley, E. J. Neural architecture search without training. *CoRR*, abs/2006.04647, 2020.

Mishkin, D. and Matas, J. All you need is a good init. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

Nguyen, T. Q. and Salazar, J. Transformers without tears: Improving the normalization of self-attention. *CoRR*, abs/1910.05895, 2019.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1310–1318, 2013.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pp. 4095–4104. PMLR, 2018.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 4780–4789, 2019.

So, D. R., Le, Q. V., and Liang, C. The evolved transformer. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 5877–5886, 2019.

Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 3645–3650, 2019.

Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1139–1147, 2013.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. S. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, 1992.

Wu, Y. and He, K. Group normalization. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, pp. 3–19, 2018.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. Nas-bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 7105–7114, 2019.

You, S., Huang, T., Yang, M., Wang, F., Qian, C., and Zhang, C. Greedynas: Towards fast one-shot NAS with greedy supernet. *CoRR*, abs/2003.11236, 2020.

Yuan, K., Ling, Q., and Yin, W. On the convergence of decentralized gradient descent. *SIAM J. Optim.*, 26(3): 1835–1854, 2016.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. Understanding and robustifying differentiable architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Zhang, H., Dauphin, Y. N., and Ma, T. Fixup initialization: Residual learning without normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Zhou, D., Zhou, X., Zhang, W., Loy, C. C., Yi, S., Zhang, X., and Ouyang, W. Econas: Finding proxies for economical neural architecture search. *CoRR*, abs/2001.01233, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

Appendix D: Proofs for Proposition 4.1

Recall we can write the dynamics of predictions as

$$\frac{d}{dt}\boldsymbol{y}^{(L)}(t) = \boldsymbol{H}(t)(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)). \tag{17}$$

By spectral theorem, we can write $\boldsymbol{H}(t) = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^T$ where $\boldsymbol{P}$ is orthogonal and $\boldsymbol{D} > 0$ is diagonal with each entry being eigenvalue of $\boldsymbol{H}(t)$. We observe that

$$\begin{aligned}
&\min_{\|\boldsymbol{x}\|=1} \boldsymbol{x}^T\boldsymbol{H}(t)\boldsymbol{x} \\
&= \min_{\|\boldsymbol{x}\|=1} (\boldsymbol{P}^T\boldsymbol{x})^T\boldsymbol{H}(t)(\boldsymbol{P}^T\boldsymbol{x}) \\
&= \min_{\|\boldsymbol{y}\|=1} \boldsymbol{y}\boldsymbol{D}\boldsymbol{y}^T \\
&= \lambda_{min}(\boldsymbol{H}(t))
\end{aligned} \tag{18}$$

Define

$$Q = \|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|_2^2. \tag{19}$$

Given Q, we can calculate the loss function dynamic as:

$$\begin{aligned}
&\frac{d}{dt}\|(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))\|_2^2 \\
&= -(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))\frac{d}{dt}\boldsymbol{y}^{(L)}(t) \\
&= -(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))\boldsymbol{H}(t)(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)) \\
&= -Q\frac{(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))^T}{\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|}\boldsymbol{H}(t)\frac{\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)}{\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|}.
\end{aligned} \tag{20}$$

Based on Eq. 18 and Eq. 19, the following inequation holds:

$$\begin{aligned}
&\frac{d}{dt}\|(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))\|_2^2 \\
&= -Q\frac{(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))^T}{\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|}\boldsymbol{H}(t)\frac{\boldsymbol{y}_i^* - \boldsymbol{y}^{(L)}(t)}{\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|} \\
&\leq -\lambda_{min}(\boldsymbol{H}(t))\|(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))\|_2^2.
\end{aligned} \tag{21}$$

Under Assumption 5.1, $\lambda_{min}(\boldsymbol{H}(t)) > 0$. Therefore, $\frac{d}{dt}\|(\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t))\|_2^2 \leq 0$ holds.

Here, we use λ_{min}, short for $\lambda_{min}(\boldsymbol{H}(t))$. The expectation of gradients of $exp(\lambda_{min}t)\|(\boldsymbol{y}^* - \boldsymbol{y}^{(L)})\|_2^2$ is

$$\begin{aligned}
&\frac{d}{dt}exp(-\lambda_{min}t)\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}\|_2^2 \\
&= -\lambda_{min}exp(-\lambda_{min}t)\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}\|_2^2 \\
&+ exp(\lambda_{min}t)\frac{d}{dt}\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|_2^2 \\
&\leq -\lambda_{min}exp(-\lambda_{min}t)\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}\|_2^2 \\
&- exp(-\lambda_{min}t)\lambda_{min}\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|_2^2 \\
&\leq 0.
\end{aligned} \tag{22}$$

Since the upper bound of the gradients is negative, $\forall t > 0$, the following inequation holds:

$$\begin{aligned}
exp(\lambda_{min}t)\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|_2^2 &\leq exp(0)\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(0)\|_2^2 \\
&\leq \|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(0)\|_2^2,
\end{aligned} \tag{23}$$

and thus,

$$\begin{aligned}
\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(t)\|_2^2 &\leq \frac{\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(0)\|_2^2}{exp(\lambda_{min}t)} \\
&\leq exp(-\lambda_{min}t)\|\boldsymbol{y}^* - \boldsymbol{y}^{(L)}(0)\|_2^2.
\end{aligned} \tag{24}$$

□