Overcoming Multi-Model Forgetting in One-Shot NAS with Diversity Maximization

Miao Zhang[1,2] Huiqi Li[1*] Shirui Pan[3] Xiaojun Chang[3] Steven Su[2]
[1]Beijing Institute of Technology [2]University of Technology Sydney [3]Monash University
Miao.Zhang-2@student.uts.edu.au, huiqili@bit.edu.cn, steven.su@uts.edu.au
{Shirui.Pan, Xiaojun.Chang}@monash.edu

Abstract

*One-Shot Neural Architecture Search (NAS) significantly improves the computational efficiency through weight sharing. However, this approach also introduces multi-model forgetting during the supernet training (architecture search phase), where the performance of previous architectures degrades when sequentially training new architectures with partially-shared weights. To overcome such catastrophic forgetting, the state-of-the-art method assumes that the shared weights are optimal when jointly optimizing a posterior probability. However, this strict assumption is not necessarily held for One-Shot NAS in practice. In this paper, we formulate the supernet training in the One-Shot NAS as a constrained optimization problem of continual learning that the learning of current architecture should not degrade the performance of previous architectures. We propose a Novelty Search based Architecture Selection (**NSAS**) loss function and demonstrate that the posterior probability could be calculated without the strict assumption when maximizing the diversity of the selected constraints. A greedy novelty search method is devised to find the most representative subset to regularize the supernet training. We apply our proposed approach to two One-Shot NAS baselines, random sampling NAS (RandomNAS) and gradient-based sampling NAS (GDAS). Extensive experiments demonstrate that our method enhances the predictive ability of the supernet in One-Shot NAS and achieves remarkable performance on CIFAR-10, CIFAR-100, and PTB with efficiency.*

1. Introduction

One-Shot Neural Architecture Search (NAS) recently attracts massive interests in automating neural network architecture design, since it can not only find state-of-the-art architectures but also significantly reduce the search hours through weight sharing. Early NAS methods adopt a nested approach to train numerous separate architectures from scratch and utilize an Evolutionary Algorithm (EA) or Reinforcement Learning (RL) to find the most promising architectures based on validation accuracy [12, 38, 29], which are highly computational-expensive and impossible for most machine learning practitioners. Several approaches were proposed to address this efficiency concern [3, 6, 34]. In particular, weight sharing, also called One-Shot NAS [4, 28], is a promising direction. One-Shot NAS defines the search space as a supernet which subsuming all candidate architectures, and the candidate architectures are evaluated through inheriting weights from the supernet. Rather than training numerous separate architectures from scratch, One-Shot NAS trains the supernet just once, thus it significantly cuts down the search cost.

One-Shot NAS relies on a critical assumption that the *validation accuracy* of architecture with inherited weights should approximate to the *test accuracy* after retraining or be highly predictive. Although Bender et al. [4] observed a strong correlation between the validation accuracy and the test accuracy when the supernet was trained through random path dropout, Sciuto et al. [30] obtained contradict results when the weights of a single path (one architecture) in the supernet were trained in each step in ENAS. This single-path training method is also the most common strategy adopted by state-of-the-art One-Shot NAS approaches [10, 19, 13, 7] and also the scenario considered in this paper. Adam et al. [1] showed that the RNN controller in One-Shot NAS does not depend on past sampled architectures, which makes its performance the same as random search. Similarly, Singh et al. [31] found that there is no visible progress in terms of the retrained performance for architectures generated by the controller during the architecture search phase in ENAS, and architectures with more shared weights usually perform worse based on the trained supernet in ENAS.

Benyahia et al. [5] defined this phenomenon as *multi-model forgetting*, which occurs when training multiple models with partially-shared weights for a single task. Suppose we are given a large supernet containing multiple models

*Corresponding Author

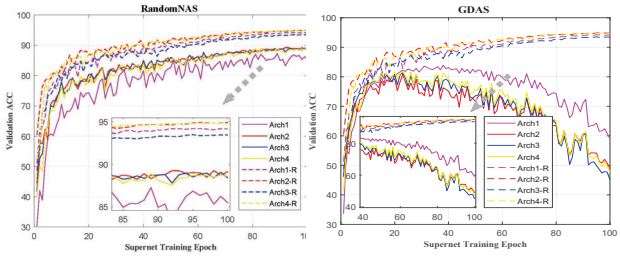


Figure 1: Validation accuracy for 4 different architectures during the supernet training for RandomNAS [19] and GDAS [11]. The solid lines ("Arch") are validation accuracy by inheriting weights from the supernet, and the dash lines ("Arch-R") are validation accuracy after retraining.

(architectures) with shared weights across them, and these models are sequentially trained on a single task, then the model with more shared weights dropped more accuracy when training another model [5, 30]. The multi-model forgetting problem is also illustrated in Fig.1, where the validation accuracy of four different architectures through inheriting weights during the supernet training is shown. It could be observed that weight sharing presents massive fluctuation in validation accuracy. Worse still, the performance of architectures by inheriting weights gets worse during the supernet training, which makes the architecture ranking based on the supernet unreliable. Apparently, although weight sharing reduces the computational hours greatly, it also introduces the multi-model forgetting in the supernet training, which will deteriorate the predictive ability of the supernet.

To overcome the multi-model forgetting in One-Shot NAS and enhance the predictive ability of the supernet, we formulate the supernet training as a constrained optimization problem of continual learning, which avoids the performance degradation of previous architectures when training a new architecture. Different from available works that only consider the performance degradation of one last architecture [5], or keep the shared parameters fixed [21], this paper tries to find the most representative subset of previous architectures to regularize the learning of current architecture during the supernet training. We develop an efficient greedy novelty search method for the constraints selection with diversity maximization, and implement our approach within two baselines, RandomNAS [19] and GDAS [11]. Experimental results demonstrates that our algorithm reduces the multi-model forgetting in their supernet training significantly. Our contributions are summarized as follows.

- Firstly, we formulate the supernet training in the One-Shot NAS as a constrained optimization problem of continual learning, where the learning of current architecture should not degrade the performance of previous architectures with partially-shared weights.
- Secondly, we devise an efficient greedy novelty search method to select the most representative constraints subset to approximate the feasible region formed by all previous architectures.
- Thirdly, the proposed approach is applied to two One-Shot NAS baselines, RandomNAS [19] and GDAS [11], to reduce the multi-model forgetting in their supernet training. Extensive experimental results illustrate the effectiveness of our method, which could reduce the multi-model forgetting and enhance the predictive ability of the supernet.

2. Background

2.1. Weight sharing Neural Architecture Search

One-Shot NAS is proposed by [28], which reduces the search time greatly through weight sharing. Different from training numerous separate architectures, One-shot NAS encodes the search space $\mathcal{A}$ as a supernet $\mathcal{W}_{\mathcal{A}}$, and the candidate neural architectures α directly inherit weights from the supernet as $\mathcal{W}_{\mathcal{A}}(\alpha)$. Since the One-Shot NAS only the supernet once in the architecture search phase, so it could greatly reduce the search time. One-Shot NAS searches for the most promising architecture α^* based the validation performance with inheriting weights from the supernet:

$$\begin{aligned} \min_{\alpha \in \mathcal{A}} \quad & \mathcal{L}_{\texttt{val}}(\mathcal{W}^*_{\mathcal{A}}(\alpha)) \\ \texttt{s.t.} \quad & \mathcal{W}^*_{\mathcal{A}}(\alpha) = \texttt{argmin}\, \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}(\alpha)) \end{aligned} \tag{1}$$

Eq. (1) is more than a challenging bilevel optimization problem, and the discrete characteristic of architecture space also makes it impossible to utilize a gradient-based method to solve it directly, where ENAS [28] uses an LSTM controller to sample architectures. Differently, [13] and [19] train the supernet based on the uniform sampling strategy to sample architectures, and a random search or an evolutionary method is adopted to find the best-performed architecture from the trained supernet.

Several state-of-the-art One-Shot methods utilize continuous relaxation to transform the discrete architecture into continuous space $\mathcal{A}_\theta$ with parameters θ to further improve the efficiency [23, 11, 33, 26]. The supernet weights and architecture parameters could be jointly optimized through:

$$(\alpha^*_\theta, \mathcal{W}_{\mathcal{A}_\theta}(\alpha^*_\theta)) = \underset{\alpha_\theta, \mathcal{W}}{\texttt{argmin}}\, \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}_\theta}(\alpha^*_\theta)) \tag{2}$$

which makes it possible to apply continuous optimizing approaches on architecture search, and the best architecture α^* could be sampled from continuous architecture representation α^*_θ.

Since Eq.(2) is supposed to train the whole supernet in each step, which has a much higher memory requirement

than ENAS, GDAS [11] further introduces a gradient-based sampler to sample single path (an architecture) in each step during the supernet training. The distribution of architectures and the supernet weight can be jointly optimized while the memory requirement equals to training a single architecture. Different from continuous relaxation, NAO [24] utilizes the LSTM based autoencoder to transform the discrete neural architectures to a continuous representations, and then perform a gradient-based method in the continuous space.

2.2. Multi-model Forgetting in One-Shot NAS

Catastrophic Forgetting is a common phenomenon in artificial general intelligence and multi-task learning, which describes that the mode usually loses the information about previous tasks after being trained on a new task [14, 18, 27]. Given a model with optimal parameters θ_A^* on dataset $\mathcal{D}_A$, its performance on $\mathcal{D}_A$ declines dramatically after this model being trained on another dataset $\mathcal{D}_B$. Methods to resolve such issues are defined as *continual learning*. Learning without forgetting (LwF) [22] adds the response of the old task as a regularization term to prevent catastrophic forgetting. Elastic weight consolidation (EWC) [17] proposes to maximize the likelihood of conditional probability $p(\theta \mid \mathcal{D})$, where $\mathcal{D}$ containing two independent data sets $\mathcal{D}_A$ and $\mathcal{D}_B$, and $\mathcal{D}_A$ is not available when trained on $\mathcal{D}_B$.

Multi-model Forgetting presents when we train multiple models in a single dataset. Different from sequentially training a model on several tasks, One-Shot NAS is supposed to apply different models, e.g., $\theta_a = (\theta_a^p, \theta^s)$ and $\theta_b = (\theta_b^p, \theta^s)$, to a single dataset $\mathcal{D}$, where θ^s is the shared weight and θ_a^p and θ_b^p are private weights. Wang et al. [32] showed that the single-modal network always outperformed the multi-modal network when training them in a single task, and the interactions between networks degraded the performance of the whole network. It is also observed in [30] and [20] that the catastrophic forgetting in the One-Shot NAS would deteriorate the performance of previous architectures after training a new architecture in the supernet. Benyahia et al. [5] defined it as the problem of *multi-model forgetting* and proposed a Weight Plasticity Loss (WPL) to reduce this forgetting in One-Shot NAS, which tries to maximize the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathcal{D})$ as:

$$\begin{aligned} p(\theta \mid \mathcal{D}) &= \frac{p(\theta_a^p, \theta_b^p, \theta^s, \mathcal{D})}{p(\mathcal{D})} = \frac{p(\theta_a^p \mid \theta_b^p, \theta^s, \mathcal{D}) p(\theta_b^p, \theta^s, \mathcal{D})}{p(\mathcal{D})} \\ &= \frac{p(\theta_a^p, \theta^s \mid \mathcal{D}) p(\mathcal{D} \mid \theta_b^p, \theta^s) p(\theta_b^p, \theta^s)}{p(\theta^s, \mathcal{D})} \\ &= \frac{p(\theta_a^p, \theta^s \mid \mathcal{D}) p(\mathcal{D} \mid \theta_b^p, \theta^s) p(\theta_b^p, \theta^s)}{\int p(\mathcal{D} \mid \theta_a^p, \theta^s) p(\theta_a^p, \theta^s) d\theta_a^p} \\ &= \frac{p(\theta_a \mid \mathcal{D}) p(\mathcal{D} \mid \theta_b) p(\theta_b)}{\int p(\mathcal{D} \mid \theta_a^p, \theta^s) p(\theta_a^p, \theta^s) d\theta_a^p} \end{aligned} \tag{3}$$

The loss function to maximize the likelihood of $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathcal{D})$ is calculated as:

$$\mathcal{L}_{WPL}(\theta_b) = \mathcal{L}_c(\theta_b) + \frac{\eta}{2}(\|\theta_b^p\|^2 + \|\theta^s\|^2) + \sum_{\theta_{s_i} \in \theta_s} \frac{\varepsilon}{2} F_{\theta_{s_i}} (\theta_{s_i} - \theta_{s_i}^*) \tag{4}$$

where $\mathcal{L}_c$ is the cross-entropy loss function, $F_{\theta_{s_i}}$ is the diagonal element of the Fisher information matrix corresponding to parameter θ_{s_i}, and is estimated by presupposing parameters (θ_a^p, θ_b^p) are independent, and θ_s^* are the shared parameters θ^s after the previous model being trained which are assumed as in the optimal points. Detailed deviation of Eq.(4) could be found in [5].

Limitations Weight Plasticity Loss (WPL) considers only one previous architecture in each step of supernet training, and it assumes the shared weights are optimal. However, the two assumptions are hard to hold in the supernet training of One-Shot NAS, since there are numerous architectures containing shared weights with current architecture and the shared weights are usually far away from optimal points. To handle these concerns, we formulate the supernet training in One-Shot NAS as a constrained optimization problem, where the learning of current architecture should not degrade the performance of previously visited architectures. We consider a subset of previous architectures as constraints to regularize the learning of current architecture, and demonstrate that the loss function of the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathcal{D})$ could be calculated without the assumption that the shared weights are optimal when maximizing the diversity of the selected architectures.

3. Methodology

3.1. Problem Formulation

One-Shot NAS sequentially trains numerous architectures and each of them is trained with few epochs. It indicates that the model weights θ_a for previous architectures are far away from optimal points, and the model weights of the current architecture are usually shared by previous architectures. Different from jointly optimizing the posterior probability as WPL [5] under the assumption that θ_a is near-optimal or keeping the shared weights fixed as Learn to Grow [21], we formulate the supernet training in the One-Shot NAS as a constrained optimization problem. In particular, we enforce the architectures inheriting weights from the supernet in current step perform better than last step with smaller training loss. Without loss of generality, we consider the typical scenario that only one architecture in the supernet is trained in each step, and the constrained optimization problem is defined as:

$$\begin{aligned} &\mathcal{W}_{\mathcal{A}}^t = \underset{\theta \in \mathcal{W}_{\mathcal{A}}(\alpha^t)}{\texttt{argmin}} \quad \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}(\alpha^t)) \\ &\texttt{s.t.} \quad \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}^t(\alpha^i)) \leq \mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}^{t-1}(\alpha^i)); \; \forall i \in \{0...t-1\} \end{aligned} \tag{5}$$

where α^t is the current architecture in step t, $\mathcal{W}_{\mathcal{A}}^t$ represents the whole weights of the supernet in in step t, and $\mathcal{W}_{\mathcal{A}}(\alpha^t)$

Algorithm 1 Greedy Novelty Search

Input: constraints archive $\mathcal{M}$, recent architectures archive $\mathcal{C}$, selected architecture α^m, n.

$N(\alpha^m, \mathcal{M}) \leftarrow$ calculate the novelty score of α^m in $\mathcal{M}$ based on Eq.(7);
for $i = 1, 2, ..., n$ **do**
  randomly sample an architecture α^r from $\mathcal{C}$;
  if $N(\alpha^r, \mathcal{M}) > N(\alpha^m, \mathcal{M})$ **then**
    replace α^m with α^r;
  end if
end for

is the weights of architecture α^t inherited from the supernet, and only $\mathcal{W}_{\mathcal{A}}(\alpha^t)$ is optimized in each step t.

3.2. Constraints Selection based on Novelty Search

The constraints in Eq.(5) prevent the learning of current architecture degrading the performance of previous architectures to overcome the multi-model forgetting in One-Shot NAS. However, the number of constraints in Eq.(5) increases linearly with the step, which makes it intractable to consider all constraints in the optimization. In practice, we try to select a subset with M constraints from previous architectures that the feasible region formed by the subset is as close to the original feasible region as possible. Intuitively, maximizing the diversity of the subset is an efficient way to find the most representative samples from the previous architectures. Based on this observation and motivated by [2], we propose a surrogate for constraint selection:

$$\begin{aligned} &\texttt{maximize}_{\mathcal{M}} \sum_{\alpha^i, \alpha^j \in \mathcal{M}} dis(\alpha^i, \alpha^j) \\ &\texttt{s.t.} \quad \mathcal{M} \subset \{\alpha^1 ... \alpha^{t-1}\}; |\mathcal{M}| = M \end{aligned} \tag{6}$$

where $dis(\alpha^i, \alpha^j)$ is to calculate the distance between architectures. Solving Eq.(6) is an NP problem, while we could use an alternative heuristic method to achieve the same goal [2]. In this paper, we proposed a greedy novelty search method to maximize the diversity of the subset. Before the archive is full, we add all newcome architectures into the subset. Once it is full, we choose the one that is most similar to the current architecture to be replaced with the one that maximizes the novelty score of the archive. We adopt a simple and standard method to measure the novelty of architectures, defined as $N(\alpha, \mathcal{M})$, which calculates the mean distance of its k-nearest neighbors in $\mathcal{M}$ from it:

$$\begin{aligned} N(\alpha, \mathcal{M}) &= \frac{1}{|S|} \sum_{\alpha^j \in S} dis(\alpha, \alpha^j) \\ S &= kNN(\alpha, \mathcal{M}) = \{\alpha^1, \alpha^2, ..., \alpha^k\} \end{aligned} \tag{7}$$

In this paper, we only measure the difference of input edges for each node in an architecture, since the order of nodes is fixed. We consider the input edges of the same node for two architectures are the same only when the two edges have same input node and same operations.

Practically, we select M constraint architectures from $|\mathcal{C}|$ recent architectures rather than all previous architectures. After obtaining M most representative constraints, we need to force the learning of the current architecture to be optimized in the feasible region formed by these constraints, and a common approach is to convert the constraints to a soft regularization loss or apply a replay buffer [2]. Algorithm 1 describes a simple implementation of our greedy novelty search method.

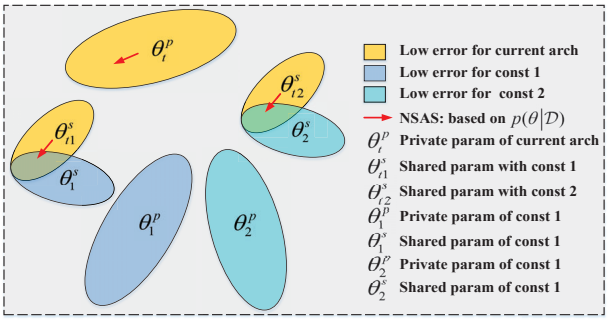


Figure 2: **NSAS** loss function ensures that the learning of current architecture will not deteriorate the performance of previous architectures in the constraint subset.

3.3. Novelty Search based Architecture Selection Loss Function

It is intractable to consider all previously visited architectures in each step of supernet training, and we select a subset of constraints $\{\alpha_1, ..., \alpha_M\}$ with diversity maximization to regularize the supernet training. The weights of these architectures in the subset are described as $\{\theta_1, ..., \theta_M\}$. The loss function for the constrained optimization problem in Eq. (5) could be described as Eq.(8) when the selected constraints are converted to a soft regularization loss:

$$\begin{aligned} \mathcal{L}_N(\mathcal{W}_{\mathcal{A}}(\alpha^t)) &= \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha^t)) + \lambda \mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha^t)) \\ &+ \beta \sum_{i=1:M} \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha^i)) + \lambda \mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha^i)) \end{aligned} \tag{8}$$

where $\mathcal{L}_{\texttt{train}}(\mathcal{W}_{\mathcal{A}}(\alpha)) = \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha)) + \lambda \mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha))$, $\mathcal{L}_c$ is the cross-entropy loss function, $\mathcal{R}$ is the ℓ_2 regularization term, β and λ are trade-offs. Here we term $\mathcal{L}_N(\mathcal{W}_{\mathcal{A}}(\alpha^t))$ as **N**ovelty **S**earch based **A**rchitecture **S**election (**NSAS**) loss function. While learning current architecture α_t, **NSAS** protects the performance of those architectures in the constraint subset by keeping these shared parameters staying in a region of low error for these constraints, as shown schematically in Fig. 2.

From Weight Plasticity Loss (WPL) to NSAS. WPL [5] regularizes the learning of current architecture by maximizing the posterior probability $p(\theta_a^p, \theta_b^p, \theta^s \mid \mathcal{D})$, where $\theta_a = \{\theta_a^p, \theta^s\}$ is the weights of the last architecture, $\theta_b = \{\theta_b^p, \theta^s\}$ is the weights of current architecture, and θ^s is their shared weights. Different from WPL which only considers one previous architecture, we consider a subset of previously visited architectures that $\theta_a = \{\theta_1, ..., \theta_M\} = \{(\theta_1^p, \theta_1^s), ..., (\theta_M^p, \theta_M^s)\}$, where θ_i^p is the private weights, and θ_i^s is the shared weights with the current architecture. When we maximize the diversity of the subset, the following two assumptions should hold true: (1) The architectures in this subset should cover all operations in the search space; (2) There are no shared weights between these architectures. Therefore, $\theta_b^p = \emptyset$ as all weights in the current architecture are shared by previous architectures, and θ_i and θ_j should be independent as we train different architecture independently. Now the posterior probability is written as:

$$\begin{aligned} p(\theta \mid \mathcal{D}) &= \frac{p(\theta_1^p ... \theta_M^p, \theta_1^s ... \theta_M^s, \mathcal{D})}{p(\mathcal{D})} = \frac{p(\theta_1 ... \theta_M, \mathcal{D})}{p(\mathcal{D})} \\ &= \frac{p(\theta_1 \mid \theta_2 ... \theta_M, \mathcal{D}) p(\theta_2 ... \theta_M, \mathcal{D})}{p(\mathcal{D})} \\ &= \prod_{i=1:M} p(\theta_i \mid \mathcal{D}) \propto \prod_{i=1:M} p(\mathcal{D} \mid \theta_i) p(\theta_i) \\ &= p(\theta) \prod_{i=1:M} p(\mathcal{D} \mid \theta_i) = p(\theta^t) \prod_{i=1:M} p(\mathcal{D} \mid \theta_i) \end{aligned} \tag{9}$$

where θ_i is the weights of architecture α^i in constraints. As only architecture α^t is trained, $p(\theta) = p(\theta^t)$, where θ^t is the weights of the current architecture α^t and θ is the all considered weights. Eq.(9) obtains the posterior probability without the assumption that θ^s in the previous step is optimal. Now the Weight Plasticity Loss could be calculated without the assumption that the shared weights are optimal when considering a subset of previously visited architectures with diversity maximization as:

$$\mathcal{L}_{WPL}(\mathcal{W}_{\mathcal{A}}(\alpha^t)) = \epsilon \mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha^t)) + \sum_{i=1:M} \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha_i)) \tag{10}$$

where ϵ is also the trade-off. And the proposed **NSAS** loss function could be also described with the Weight Plasticity Loss in Eq.(10) as:

$$\begin{aligned} \mathcal{L}_N(\mathcal{W}_{\mathcal{A}}(\alpha^t)) &= \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha^t)) + \lambda \mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha^t)) \\ &\quad + \beta \sum_{i=1:M} \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha^i)) + \lambda \mathcal{R}(\mathcal{W}_{\mathcal{A}}(\alpha^i)) \\ &= \mathcal{L}_c(\mathcal{W}_{\mathcal{A}}(\alpha^t)) + \beta \mathcal{L}_{WPL}(\mathcal{W}_{\mathcal{A}}(\alpha^t)) \end{aligned} \tag{11}$$

From Eq.(11), we can find that our proposed loss function is attempted to not only optimize the WPL but also optimize the learning of current architecture (also the shared weights). That is because the shared weights are usually far from the optimal point in One-Shot NAS, and we should not only overcome the forgetting, but also optimize the shared weights towards the optimal point.

Algorithm 2 One-Shot NAS-NSAS

Input: $\mathcal{D}_{train}$, $\mathcal{D}_{val}$, $\mathcal{W}$, constraints archive $\mathcal{M} = \emptyset$, M, batch size b, supernet training iteration T

for $t = 1, 2, ..., (T * \texttt{size}(\mathcal{D}_{train})/b)$ **do**
  if $\texttt{size}(\mathcal{M}) < M$ **then**
    sample α^t based on gradient search or random search, and update the weights $\mathcal{W}_A(\alpha)$ by normal loss function, and add architecture α into $\mathcal{M}$;
  else
    sample α^t based on gradient search or random search, select the architecture α^m that is most similar to α^t from $\mathcal{M}$, and replace α^m with α^r to maximize the diversity of $\mathcal{M}$ based on Algorithm 1. Update the weights $\mathcal{W}_A(\alpha)$ by our proposed loss function in Eq.(8) or a replay buffer;
  end if
end for
Obtain α^* based on Eq.(1) (RandomNAS-NASA) or Eq.(2) (GDAS-NASA).

3.4. One-Shot NAS with Novelty Search based Architecture Selection

Our loss function is applied to two popular One-Shot NAS: RandomNAS [19] and GDAS [11]. Same as the most common weight sharing NAS, we only train a single path in each step in the architecture search phase. It is easy to incorporate our proposed loss function to random sampling based NAS (RandomNAS) as it also trains a single path in each step. However, most gradient-based NAS methods, like DARTS [23] and SNAS [33], train the whole supernet in each step during the supernet training, which violates the assumption in this paper. In this paper, we adopt GDAS [11] as the gradient-based sampling NAS baseline, which utilizes the Gumbel-Max trick [15, 25, 33] to relax the discrete architecture distribution to be continuous and differentiable. The **argmax** function is applied to the re-parameterized architecture distribution, to sample an architecture in each step of the supernet training during the forward pass. The **softmax** function is adopted during the backward pass for architecture learning. Algorithm 2 presents the One-Shot NAS with our proposed **NSAS** loss function, termed as One-Shot NAS-NSAS.

4. Experiments and Results

To evaluate the effectiveness of our proposed algorithm, we apply our method to both RandomNAS [19] and GDAS [11] on datasets CIFAR-10, CIFAR-100, and Penn Treebank (PTB). All experimental designs are following the settings in [19, 23] for a fair comparison. Our new methods are denoted as **RandomNAS-NSAS** and **GDAS-NSAS**. We compare our methods with the state-of-the-art One-Shot

Method	Test Error (%)		Parameters	Search Cost	Memory	Supernet
	CIFAR-10	CIFAR-100	(M)	(GPU Days)	Consumption	Optimization
NAO-WS [24]	3.53	-	2.5	-	Single path	Gradient
ENAS [28]	2.89	18.91† [11]	4.6	0.5	Single path	RL
SNAS [33]	2.85±0.02	20.09*	2.8	1.5	Whole Supernet	Gradient
PARSEC [8]	2.86±0.06	-	3.6	0.6	Single path	Gradient
BayesNAS [37]	2.81±0.04	-	3.40	0.2	Whole Supernet	Gradient
RENAS [9]	2.88±0.02	-	3.5	6	-	RL&EA
MdeNAS [36]	2.40	-	4.06	0.16	Single path	MDL
MdeNAS* [36]	2.87*	17.61*	3.78*	0.16	Single path	MDL
DSO-NAS [35]	2.84±0.07	-	3.0	1	Whole Superne	Gradient
WPL [5]	3.81	-	-	-	Single path	RL
Random baseline [23]	3.29±0.15	-	3.2	4	-	Random
DARTS (1st) [23]	2.94	-	2.9	1.5	Whole Supernet	Gradient
DARTS (2nd) [23]	2.76±0.09	17.54 †[11]	3.4	4	Whole Supernet	Gradient
RandomNAS [19]	2.85±0.08	17.63*	4.3*	2.7	Single path	Random
RandomNAS-NSAS	**2.64(2.50)**	17.56(**16.85**)	3.08	0.7	Single path	Random
GDAS [11]	2.93	18.38	3.4	0.21	Single path	Gradient
GDAS-NSAS	2.73	18.02	3.54	0.4	Single path	Gradient

Table 1: Test errors on CIFAR-10, compared with state-of-the-art NAS approaches. "*" indicates the results reproduced based on the best reported cell structures with the same experimental setting as ours. "†" indicates the results are reported in the [11]. We do not reproduce those methods with "-" in CIFAR-100 experiment since they are with different search spaces or do not report their best structures. All models are trained with 600 epochs, and we also train our best found model (RandomNAS-NSAS) with more epochs (1000 training epochs) to get the state-of-the-art results. The best models obtained by all One-Shot NAS methods are trained with cutout.

NAS methods and evaluate the supernet predictive ability of our approach compared with baselines.

4.1. Architecture Search for Convolutional Cells

We conduct comparative experiments for convolutional neural architecture search on CIFAR-10. The search space and hyperparameters setting are following the settings in [23, 19] for a fair comparison. This search space searches for micro-cell structures, which are stacked in series to form the final structure. In the supernet training (architecture search) stage, we only stack 8 cells to build the supernet with 16 initial channels and 64 batch size. After supernet training and obtaining the promising cells, we stack 20 cells to form the final architecture, and train it with 96 batch size. The comparison results are demonstrated in Table 1, which can be summarized as follows:

- Compared with RandomNAS and GDAS which both employ normal cross-entropy loss function, our proposed NSAS loss function could greatly enhance the search results, where RandomNAS obtains 5.6% improvement and 4.8% improvement for GDAS. These results also demonstrate that the effectiveness of the proposed loss function, which could relieve the rank disorder incurred by weight sharing and improve the supernet predictive ability.
- Compared with other NAS methods, our RandomNAS-NSAS achieves a competitive result with a 2.64% test error on CIFAR-10, and a 2.50% test error with 1000 training epochs. This is an inspiring result to validate our design to overcoming multi-model forgetting.
- Since the proposed method needs to evaluate more architectures during the supernet training, it has a little bit higher search cost than the baselines. However, the proposed method is still very efficient that the supernet training only costs 0.7 GPU day for RandomNAS-NSAS and 0.4 GPU day for GDAS-NSAS. We further transfer the found architecture on CIFAR-10 to CIFAR-100 to evaluate its transferability.

The architecture evaluation setting on CIFAR-100 is the same as CIFAR-10, and all results are reported in Table 1. We also increase the number of initial filters to 50 (and the parameters increase to 5.8 M) before transfer our best found structure to CIFAR-100 to enhance its performance, and we could observe that our network achieves state-of-the-art results (a test error of 16.85%) among all compared methods.

Method	Perplexity		Paras	Search
	Valid	Test	(M)	Cost
ENAS* [19]	60.8	58.6	24	0.5
NAO-WS [24]	-	56.6	27	0.4
WPL [5]	-	61.9	-	-
Random baseline [23]	61.8	59.4	23	2
DARTS (1st) [23]	60.2	57.6	23	0.5
DARTS (2nd) [23]	58.8	56.6	23	1
DARTS (2nd)* [23]	59.21*	56.71*	23	1
RandomNAS [19]	57.8	55.5	23	0.25
RandomNAS* [19]	59.7*	57.16*	23	0.25
RandomNAS-NSAS	**59.22**	**56.84**	23	0.62
GDAS [11]	59.8	57.5	23	0.4
GDAS* [11]	60.23*	57.69*	23	0.4
GDAS-NSAS	59.74	57.24	23	0.50

Table 2: Comparison results with One-Shot NAS methods on PTB. "*" indicates the results reproduced with the same experimental setting as ours.

4.2. Architecture Search for Recurrent Cells

We also conduct experiments for the RNN cell structure search on the PTB dataset, and the search space and hyperparameters setting are following settings in [19, 23] for fair comparison. In the RNN supernet training, the hidden and the embedding sizes are set as 300, and the supernet is trained for 300 epochs with batch size 128. After supernet training and obtaining the promising cells, we change the hidden and the embedding sizes to 850, and train the network for 3600 epochs with 64 batch size. The comparison results on PTB with other baselines are presented in Table 2. The model discovered by our Random-NSAS achieves a validation perplexity of 59.22 and a test perplexity of 56.84, which is on par with the state-of-the-art approaches. A further observation on our proposed loss function shows the effectiveness of our NSAS loss function when it is applied to two baselines. RandomNAS-NSAS and GDAS-NSAS both surpass the original baselines with normal loss function. These results again show that the NSAS loss function could enhance the supernet predictive ability, as evidenced by the fact that our RandomNAS-NSA and GDAS-NSAS both beat the two baselines with the normal loss function.

4.3. Supernet Predictive Ability Comparison

Multi-model Forgetting in One-Shot NAS To demonstrate catastrophic forgetting in neural architecture search, we conduct experiments on convolutional cell search to present the differences between weight sharing and retraining based architecture ranking for two baselines, RandomNAS and GDAS. We tracked the validation accuracy of

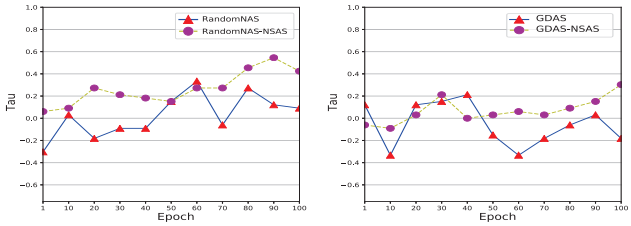
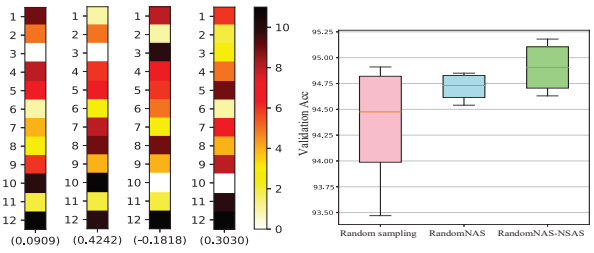


Figure 3: Kendall Tau metric (τ) of architecture ranking based on weight sharing and retraining .

inheriting weights for several fixed sampled architectures with 8 stacked cells, and also plot the validation accuracy for 100 epochs when retraining these separate architectures from scratch in Figure 1. We could observe that architectures directly inherit weights from the supernet present massive fluctuation in their validation accuracy, which makes it hard to identify the quality of architecture according to this accuracy. What is worse, the architecture ranking results completely violate the primary hypothesis in weight sharing NAS that architectures with higher validation performance based on weight sharing should achieve better retraining performance. It should be noticed from Fig. 1 (b) that, the performance of architectures by inheriting weights even gets worse during the supernet training in GDAS.

Supernet Predictive Ability Comparison To verify the effectiveness of our approach in relieving the rank disorder in weight sharing neural architecture search, we incorporate our proposed loss function into the RandomNAS and GDAS frameworks, and conduct more experiments on the architecture ranking prediction. The validation accuracy through inheriting supernet weights is usually different from retraining, while it should at least be highly predictive, especially for those architectures with excellent performance. We sample 4 excellent architectures in 4 rounds based on RandomNAS and RandomNAS-NSAS, respectively. We individually train these 12 architectures from scratch (4 from RandomNAS, 4 from RandomNAS-NSAS, and 4 randomly sampled in the previous experiment), and calculate the correlation of architecture ranking between the validation accuracy through weight sharing and retraining. Figure 3 presents the Kendall Tau (τ) metric [16, 36] of architecture ranking based on weight sharing and retraining during the supernet training, which depicts the rank difference of normal cross-entropy loss function and our proposed loss function. Figure 4 (a) gives the final Kendall Tau (τ) metric values for RandomNAS and GDAS with different loss functions after supernet training. It could be observed that the normal loss function presents poor supernet predictive ability, which obtains $\tau = 0.0909$ and $\tau = -0.1818$ for



(a) Architectures ranking difference after supernet training (b) Retraining validation accuracy

Figure 4: (a) Architectures ranking difference between the retraining and inheriting weights from trained supernet for RandomNAS, RandomNAS-NSAS, GDAS, and GDAS-NSAS, respectively. (b) The mean retraining validation accuracy for architectures obtained through different methods.

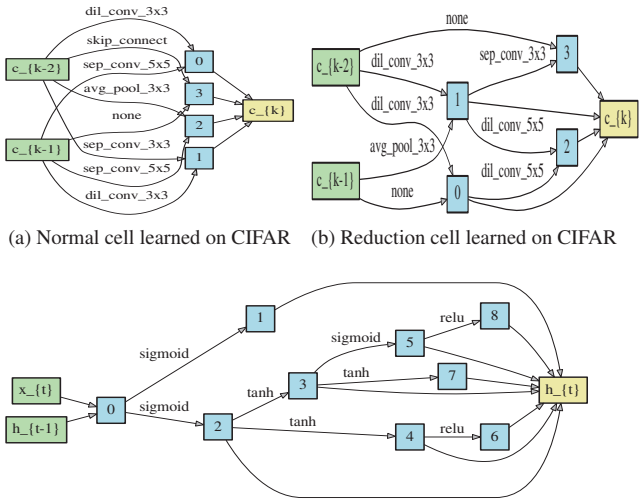
RandomNAS and GDAS, respectively. Although the supernet trained by NSAS is not able to obtain the precisely identical architecture rank, our proposed loss function achieves a positive correlation with much better Kendall Tau metric ($\tau = 0.4242$ and $\tau = 0.3030$ for RandomNAS-NSAS and GDAS-NSAS, respectively). A supernet with better predictive ability tends to obtain architectures with better retraining performance, and Fig.4 (b) plots the mean retraining validation accuracy of sampled architectures through different methods. We could find that RandomNAS-NSAS achieves better results than normal RandomNAS, which further verifies the effectiveness of our proposed method.

4.4. Discussion

We conduct a series of architecture search experiments with two One-Shot baselines, and RandomNAS tends to achieve better performance than GDAS (in both the normal and the proposed loss function). One underlying reason may be the gradient-based methods usually obtain the local optimal solution once the supernet is trained, while RandomNAS further conducts the model selection (using random search or EA) to find the global optimal solution from the trained supernet. Since RandomNAS needs to evaluate numerous architectures base on the trained supernet, it usually has a higher search cost than GDAS.

Fig. 5 visualizes the best-found cells by the proposed approach for CNN and RNN models, and the codes and trained models could be found here [1]. After revisiting the experimental results in previous subsections, it is clear that the proposed loss function could greatly enhance the predictive ability of the supernet, which greatly improves the performance of the found architectures for two NAS baselines, RandomNAS and GDAS. The supernet training in

[1]https://github.com/MiaoZhang0525/NSAS_FOR_CVPR.



(a) Normal cell learned on CIFAR (b) Reduction cell learned on CIFAR

(c) Reccurent cell learned on PTB

Figure 5: The best cells discovered on CIFAR-10 and PTB.

One-Shot NAS is definitely a multi-task problem, and devising a more appropriate loss function rather than using the normal one is a promising direction to improve the performance of One-Shot NAS methods.

5. Conclusion and Future Works

This paper originally formulates the supernet training as a constrained optimization problem to relieve the multi-model forgetting in One-Shot neural architecture search. A greedy novelty search method is adopted to select a representative subset of constraints to regularize the supernet training in the feasible region, and a **N**ovelty **S**earch based **A**rchitecture **S**election (NSAS) loss function is accordingly devised to overcome the multi-model forgetting. We incorporate the proposed loss function into two One-Shot NAS baselines. Experimental results on the neural architecture search show the effectiveness of the proposed method. In particular, our method improves supernet predictive ability and achieves excellent results in both convolutional cell search and recurrent cell search. In the future work, we will focus on searching on a latent space through transforming the discrete architectures into continuous representations, and also leveraging human knowledge on DNN design to search for architectures with better transferable ability.

6. Acknowledgment

This work is supported in part by NSFC grant 61702415, Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA) under grant no. DE190100626, Air Force Research Laboratory and DARPA under agreement number FA8750- 19-2-0501.

References

[1] George Adam and Jonathan Lorraine. Understanding neural architecture search techniques. *arXiv preprint arXiv:1904.00438*, 2019.

[2] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[3] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. 2018.

[4] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning (ICML)*, pages 549–558, 2018.

[5] Yassine Benyahia, Kaicheng Yu, Kamil Bennani Smires, Martin Jaggi, Anthony C Davison, Mathieu Salzmann, and Claudiu Musat. Overcoming multi-model forgetting. In *International Conference on Machine Learning (ICML)*, pages 594–603, 2019.

[6] Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J Weston. Smash: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2018.

[7] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*, 2019.

[8] Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019.

[9] Yukang Chen, Gaofeng Meng, Qian Zhang, Shiming Xiang, Chang Huang, Lisen Mu, and Xinggang Wang. Renas: Reinforced evolutionary neural architecture search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[10] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019.

[11] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[12] Minghao Guo, Zhao Zhong, Wei Wu, Dahua Lin, and Junjie Yan. Irlas: Inverse reinforcement learning for architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[13] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1906.07590*, 2019.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[15] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

[16] Maurice G Kendall. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.

[17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[18] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4652–4662, 2017.

[19] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *The Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

[20] Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot nas by suppressing the posterior fading. *arXiv preprint arXiv:1910.02543*, 2019.

[21] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *International Conference on Machine Learning (ICML)*, pages 3925–3934, 2019.

[22] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

[23] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[24] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7816–7827, 2018.

[25] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning RepresentationsV (ICLR)*, 2017.

[26] Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik-Manor. Xnas: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[27] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

[28] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, pages 4092–4101, 2018.

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[30] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

[31] Prabhant Singh, Tobias Jacobs, Sebastien Nicolas, and Mischa Schmidt. A study of the learning progress in neural architecture search techniques. *arXiv preprint arXiv:1906.07590*, 2019.

[32] Weiyao Wang, Du Tran, and Matt Feiszli. What makes training multi-modal networks hard? *arXiv preprint arXiv:1905.12681*, 2019.

[33] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[34] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[35] Xinbang Zhang, Zehao Huang, and Naiyan Wang. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567*, 2019.

[36] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial distribution learning for effective neural architecture search. In *International Conference on Computer Vision (ICCV)*, 2019.

[37] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *International Conference on Machine Learning (ICML)*, 2019.

[38] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710, 2018.