Evaluating the Search Phase of Neural Architecture Search

Christian Sciuto [* 1] **Kaicheng Yu** [* 2] **Martin Jaggi** [3] **Claudiu Musat** [1] **Mathieu Salzmann** [2]

Abstract

Neural Architecture Search (NAS) aims to facilitate the design of deep networks for new tasks. Existing techniques rely on two stages: searching over the architecture space and validating the best architecture. Evaluating NAS algorithms is currently solely done by comparing their results on the downstream task. While intuitive, this fails to explicitly evaluate the effectiveness of their search strategies.

In this paper, we extend the NAS evaluation procedure to include the search phase. To this end, we compare the quality of the solutions obtained by NAS search policies with that of random architecture selection. We find that: (i) On average, the random policy outperforms state-of-the-art NAS algorithms; and (ii) The results and candidate rankings of NAS algorithms do not reflect the true performance of the candidate architectures. While our former finding illustrates the fact that the NAS search space has been sufficiently constrained so that random solutions yield good results, we trace the latter back to the weight sharing strategy used by state-of-the-art NAS methods. In contrast with common belief, weight sharing negatively impacts the training of good architectures, thus reducing the effectiveness of the search process. We believe that following our evaluation framework will be key to designing NAS strategies that truly discover superior architectures.

1. Introduction

By automating the design of a neural network for the task at hand, Neural Architecture Search (NAS) has tremendous potential to impact the practicality of deep learning (Zoph & Le, 2017; Liu et al., 2018b;a; Tan et al., 2018; Baker et al., 2016). This is part of a larger trend – automated machine learning (AutoML) – that promises to solve or at least alleviate the scarcity of ML experts needed to design custom architectures. A typical NAS technique (Zoph & Le, 2017; Pham et al., 2018; Liu et al., 2018a) has two stages: the search phase, which aims to find a good architecture, and the validation one, where the best architecture is trained from scratch and evaluated on the test data.

Table 1. **Typical evaluation of NAS algorithms vs evaluation in this paper.** Existing works typically do not perform extensive comparisons to randomly sampled architectures and, in most cases, do not report the results of their algorithms with different random seeds. Here, we show that a fair and thorough comparison to random architecture selection, using multiple random seeds, is crucial to analyze the effectiveness of search policies. As shown in the last column, where we report the mean validation perplexity on the Penn Tree Bank dataset, our evaluation reveals that the state-of-the-art NAS algorithms tend to perform at best on par with random search, emphasizing the need for new search policies.

	Comp. to random	Multiple seeds	Mean valid PPL
ENAS	✗	✗	61.91 ± 0.50
DARTS	✗	✓	63.29 ± 1.18
NAO	✗	✗	63.22 ± 1.12
Random (Ours)	✓	✓	62.50 ± 0.86

The evaluation of NAS techniques is not trivial. As the resulting networks are only as good as the results they obtain, the standard strategy (Zoph & Le, 2017; Pham et al., 2018; Liu et al., 2019b; 2018a), focuses on their behavior on downstream tasks, that is, the results on the test data. While this may seem intuitive, it fails to provide a complete picture of these algorithms; the effectiveness of the search procedure is never directly evaluated.

In this paper, we introduce a framework to evaluate the search phase of NAS algorithms. Our investigation aims to improve the understanding of what causes the state-of-the-art results obtained by networks generated by NAS. In particular, to facilitate the search procedure, state-of-the-art NAS techniques, such as DARTS (Liu et al., 2019b), NAO (Luo et al., 2018) and ENAS (Pham et al., 2018), rely on two approximations: a reduced search space and weight sharing across different architectures. The impact of these schemes cannot be analyzed from the final performance; doing so requires evaluating the search process.

[*]Equal contribution [1]Artificial Intellegence Lab, Swisscom [2]Computer vision lab, EPFL [3]Machine learning and optimization lab, EPFL. Correspondence to: Christian Sciuto <sciutochristian@gmail.com>, Kaicheng Yu <kaicheng.yu@epfl.ch>.

To this end, we therefore propose to compare the quality of the NAS solutions with a random policy, which uniformly randomly samples an architecture from the same search space as the NAS algorithms, and trains it using the same hyper-parameters as the NAS solutions. To reduce randomness, the search using each policy is repeated several times, always changing the starting random seed. The results of this comparison for the state-of-the-art NAS algorithms DARTS (Liu et al., 2019b), NAO (Luo et al., 2018) and ENAS (Pham et al., 2018) are surprising:

1. On average, as shown in Table 1, random selection is not just competitive with but outperforms these techniques. We identify two reasons behind this. The first one is that the search space of these NAS algorithms has been sufficiently constrained during its construction so that even a random architecture in this space provides good results. The second is directly related to our second finding below.

2. Weight sharing not only harms the individual networks' performance but also drastically shuffles the ranking of the different architectures, compared to the true ranking by quality. We evidence this by performing experiments on a reduced search space for which we can train all architectures exhaustively, without weight sharing. On the Penn Tree Bank (PTB) dataset (Marcus et al., 1994a), we show that the architecture rankings obtained with and without weight sharing are entirely different, to the point of being uncorrelated, as indicated by a Kendall Tau value of -0.004 on average over 10 different runs. Moreover, none of the studied methods find the best solution in this small space.

In other words, we disprove the common belief that the quality of architectures trained with weight sharing is similar to that of architectures trained without. We show that the difference in ranking negatively impacts the search phase of NAS algorithms, which, in addition to sensitivity to different seeds, seriously impedes NAS robustness and performance.

In short, this paper constitutes the first attempt at evaluating the search phase of NAS. This allowed us to identify two key characteristics of state-of-the-art NAS algorithms: The importance of the search space and the negative impact of weight sharing. We show that, due to weight sharing, the search results become decoupled from the final test results. We believe this evaluation framework will be instrumental in designing NAS search strategies that are superior to the random one. We will make our code publicly available upon acceptance of this paper.

2. Related Work

When initially proposed by Zoph & Le (2017), neural architecture search (NAS) demonstrated great potential to surpass the human design of deep networks for both visual recognition (Liu et al., 2018b; Ahmed & Torresani, 2018; Chen et al., 2018; Pérez-Rúa et al., 2018; Liu et al., 2019a) and natural language processing (Zoph & Le, 2017; Pham et al., 2018; Luo et al., 2018; Zoph et al., 2018; Liu et al., 2018b; Cai et al., 2018). To achieve this, NAS relies on defining a search space as a directed acyclic graph (DAG), where a node represents a parametric operation and an edge an operation with multiple choices (Zoph & Le, 2017). The search is then performed by sampling a sub-graph, training it to convergence on the target dataset, and updating the sub-graph sampling strategy. Such strategies include reinforcement learning samplers (Zoph & Le, 2017; Zoph et al., 2018; Pham et al., 2018), evolutionary algorithms (Xie & Yuille, 2017; Real et al., 2017; Miikkulainen et al., 2019; Liu et al., 2018b; Lu et al., 2018) and performance predictors (Liu et al., 2018a; Luo et al., 2018).

Neural architecture search with weight sharing. The potential of vanilla NAS, however, comes with the drawback of requiring thousands of GPU hours even for small datasets, such as PTB (Marcus et al., 1994b) and CIFAR-10 (Krizhevsky et al., 2009). Furthermore, even when using such heavy computational resources, vanilla NAS had to restrict the number of trained architectures from a total of $\mathsf{O}(\mathsf{n}10^9)$, with n the number of nodes in the DAG, to $\mathsf{O}(\mathsf{n}10^4)$, and increasing the sampler accuracy can only be achieved by increasing the resources. Weight sharing therefore quickly emerged as a solution to make NAS practical.

Pham et al. (2018) were the first to propose sharing parameters across different sampled models. By preventing having to train each model from scratch, but by instead starting from the previously-trained models, weight sharing reduces the search space to $\mathsf{O}(\mathsf{n})$ and the resource requirements to a few GPU days (Pham et al., 2018; Luo et al., 2018; Liu et al., 2018a; 2019a; Chen et al., 2018). Variations on this approach were then introduced to replace the reinforcement learning sampler with gradient descent (Liu et al., 2019b) and to further rely on a variational auto-encoder instead of a vector-based architecture representation (Luo et al., 2018). Altogether, these three weight-sharing-based strategies, ENAS (Pham et al., 2018), DARTS (Liu et al., 2019b) and NAO (Luo et al., 2018) constitute the state of the art in neural architecture search. The quality of these methods, however, is only evaluated on the downstream task, that is, after training the best architecture from scratch on the target dataset, in most cases using different hyper-parameters than those employed during the search phase.

Evaluation of NAS algorithms. Liu et al. (2019b) took the important step of evaluating the robustness of the test performance for search results obtained with different random seeds. This is an exception: Competing methods do not examine this issue. Here, we aim to further the understanding of the mechanisms behind the search phase of NAS algorithms. Specifically, we propose doing so by comparing them with a simple random search policy, which uniformly randomly samples one architecture per epoch in the same search space as the NAS techniques.

While some works have provided partial comparisons to random search, these comparisons unfortunately did not give a fair chance to the random policy. Specifically, Pham et al. (2018) report the results of only a single random architecture, and Liu et al. (2018b) those of an architecture selected among 8 randomly sampled ones as the most promising one after training for 300 epochs only. Here, we show that a fair comparison to the random policy, obtained by training all architectures for 1000 epochs and averaging over multiple random seeds to compute a reliable evaluation, yields a different picture; the state-of-the-art search policies are no better than the random one.

The reason behind this comparison is the observation that there is only a weak correlation between the performance of the searched architectures and the ones trained from scratch during the evaluation phase. This phenomenon was already observed by Zela et al. (2018) but the analysis of its impact or its causes went no further. Here, by contrast, we link this difference in performance between the search and evaluation phases to the use of weight sharing.

The impact of weight sharing was analyzed by Bender et al. (2018) in the context of their approach by studying the influence of their dropout strategy on the correlation between architectures trained with and without weight sharing. Our work differs from (Bender et al., 2018) in two fundamental ways. Firstly, instead of limiting ourselves to sampling architectures from the search space, we make use of a reduced space with two nodes where we can perform a complete evaluation of all architectures. Secondly, and more importantly, we do not specifically focus on weight sharing, but rather analyze it to understand the consequences its use has on the search phase of NAS algorithms.

3. Evaluation Framework

In this section, we detail our evaluation framework for the NAS search phase. As depicted in Fig. 1(a,b), typical NAS algorithms consist of two phases:

- **Search:** The goal of this phase is to find the best candidate architecture from the search space. This is where existing algorithms, such as ENAS, DARTS and NAO, differ; they each correspond to a different search algorithm. Nevertheless, for all the algorithms, the search depends heavily on initialization. In all the studied policies, initialization is random and the outcome thus depends on the chosen random seed.

- **Evaluation:** In this phase, all the studied algorithms retrain the best model found in the search phase. The retrained model is then evaluated on the test data.

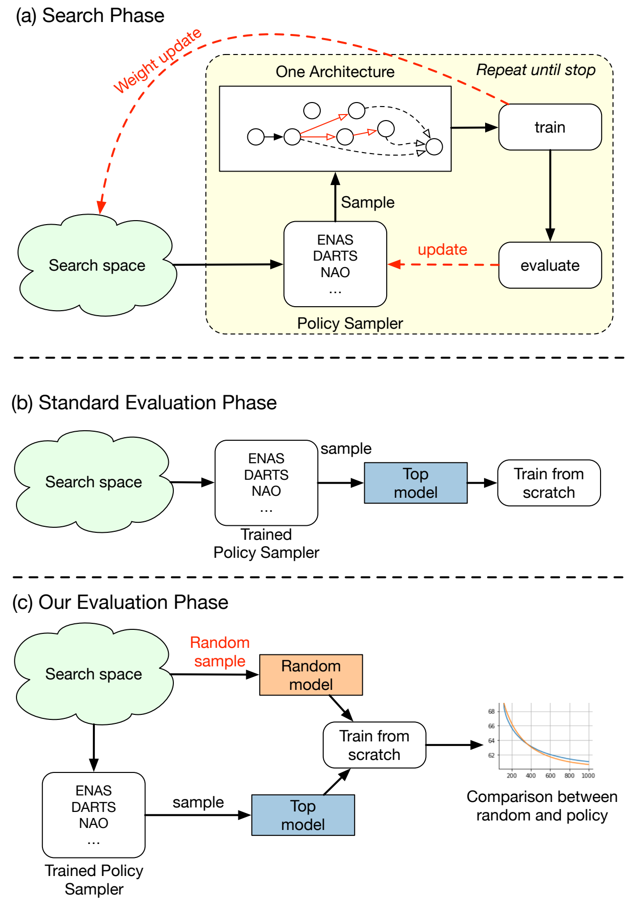

Figure 1. **Evaluating neural architecture search.** Existing frameworks consist of two phases: **(a)** The search phase, where a sampler is trained until convergence or a pre-defined stopping criterion; **(b)** The evaluation phase that trains the best model from scratch and evaluates it on the test data. Here, we argue that one should evaluate the search itself. To this end, as shown in **(c)**, we compare the best architecture found by the NAS policy with *a single* uniformly randomly sampled architecture. For this comparison to be meaningful, we repeat it with different random seeds for both training the NAS sampler and our random search policy. We then report the mean and standard deviations over the different seeds.

3.1. Evaluating the NAS Search

As argued above, the standard evaluation of NAS techniques focuses solely on the final results on the test data. Here, by contrast, we aim to evaluate the search phase itself, which is the phase that truly differentiates existing algorithms.

To do this, as illustrated in Fig. 1(c), we first establish a baseline. We propose to compare the search phase of existing algorithms with a random search policy. An effective search algorithm should yield a solution that clearly outperforms the random policy. In the remainder of this section,

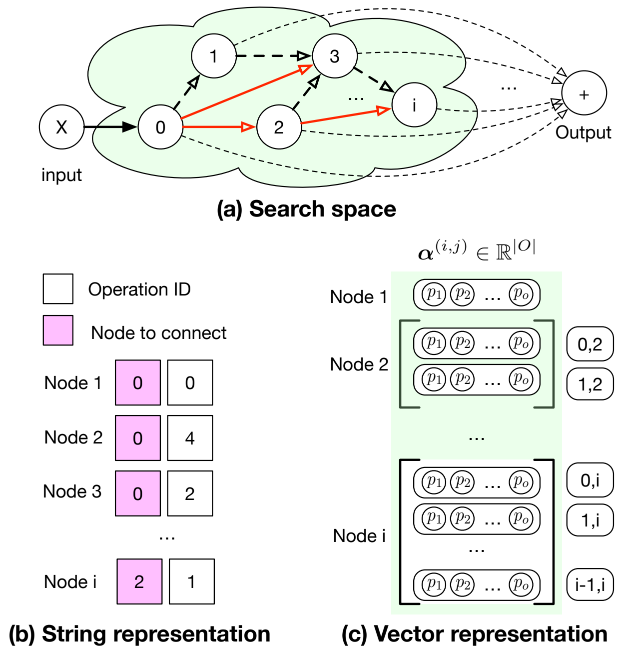

Figure 2. **Search space of NAS algorithms.** Typically, the search space is encoded as **(a)** a directed acyclic graph, and an architecture can be represented as **(b)** a string listing the node ID that each node is connected to, or the operation ID employed by each node. **(c)** An alternatively representation is a list of vectors α of size $\frac{n(n+1)}{2}|\mathcal{O}|$, where n is the number of nodes and $\mathcal{O}$ is the set of all operations. Each vector, $\alpha^{(i,j)}$, captures, via a softmax, the probability p_o that operation o is employed between node i and j. Note that any node only takes one incoming edge.

we introduce our framework to compare NAS search algorithms with random search and then discuss the specific NAS algorithms we evaluated.

3.1.1. NAS Search Space Representations

Following common practice in NAS (Zoph & Le, 2017), our starting point is a neural search space for a recurrent architecture, as illustrated in Fig. 2. A candidate architecture sampled from this space connects the input and the output nodes through a sequence of intermediary ones. Each node is connected to others and has an operation attached to it.

A way of representing this search space (Pham et al., 2018; Luo et al., 2018), depicted in Figure 2(b), is by using strings. Each character in the string indicates either the node ID that the current node is connected to, or the operation selected for current node. Operations include the identity, sigmoid, tanh and ReLU (Nair & Hinton, 2010).

Following the alternative way introduced in (Liu et al., 2019b), we make use of a vectorized representation of these strings. More specifically, as illustrated by Fig. 2(c), a node ID, resp. an operation, is encoded as a vector of probabilities over all node IDs, resp. all operations. For instance, the connection between nodes i and j is represented as $y^{(i,j)}(x) = \sum_{o \in \mathcal{O}} p_o o(x)$, with $\mathcal{O}$ the set of all operations, and $p_o = \texttt{softmax}(\ _o) = \exp(\ _o){=}\sum_{o' \in \mathcal{O}} \exp(\ _{o'})$ the probability of each operation.

3.1.2. Comparing to Random Search

We implement our random search policy by simply assigning uniform probabilities to all operations. That is, for each node j , we set $\ _o^{(i,j)} = \ _{o'}^{(i,j)}$, $\forall$i < j; o; o $'$ $\in \mathcal{O}$. We then randomly sample a connection to *one* previous node from the resulting distributions.

An effective search policy should outperform the random one. We therefore compare it with each studied NAS algorithm. To this end, as illustrated in Fig. 1(c), we compute the validation results of the best architecture found by the NAS algorithm trained from scratch, as well as those of *a single* randomly sampled architecture.

Comparing these values for a single random seed would of course not provide a reliable measure. Therefore, we repeat this process for multiple random seeds used both during the search phase of the NAS algorithm and to sample a random architecture as described above. We then report the means and standard deviations of these results over the different seeds. Note that while we use different seeds for the search and random sampling, we always use the same seed when training the models from scratch during the evaluation phase.

Our use of multiple random seeds and of the same number of epochs for the NAS algorithm and for our random search policy makes the comparison fair. This contrasts with the comparisons performed in (Pham et al., 2018), where the results of only a single random architecture were reported, and in (Liu et al., 2019b), which selected a single best random architecture among an initial set of 8 after training for 300 epochs only.

3.2. Evaluated NAS Algorithms

To show the generality of our evaluation framework, we compare our random search policy to three state-of-the-art neural architecture search algorithms: DARTS (Liu et al., 2019b), NAO (Luo et al., 2018) and ENAS (Pham et al., 2018). These three systems, discussed below, essentially only differ in terms of sampling logic.

ENAS adopts a reinforcement learning sampling strategy that is updated with the REINFORCE algorithm. The sampler is implemented as a two-layer LSTM (Hochreiter & Schmidhuber, 1997) and generates a sequence of strings, each character of which represents either a previous node

ID that the current node is connected to, or an operation. In the training process, each candidate sampled by the ENAS controller is trained on an individual mini-batch. At the end of each epoch, the controller samples new architectures that are evaluated on a single batch of the validation dataset. After this, the controller is updated accordingly using these validation metrics.

DARTS vectorizes the aforementioned strings as discussed in Section 3.1.1 and shown in Fig. 2(c). The sampling process is then parameterized by the vector , which is optimized via gradient-descent in a dual optimization scheme: The architecture is first trained while fixing , and is then updated while the network is fixed. This process is repeated in an alternating manner. In the evaluation phase, DARTS samples the top-performing architecture by using the trained vector as probability prior, i.e., the final model is not a soft average of all paths but one path in the DAG, which makes its evaluation identical to that of the other NAS algorithms.

NAO implements a gradient-based algorithm, but instead of vectorizing the strings as in DARTS, it makes use of a variational auto-encoder (VAE) to learn a latent representation of the candidate architectures. Furthermore, it uses a performance predictor, which takes a latent vector as input to predict the corresponding architecture performance. In short, the search phase of NAO consists of first randomly sampling an initial pool of architectures and training them, so as to obtain a ranking. This ranking is then used to train the encoder-predictor-decoder network, from which new candidates are sampled, and the process is repeated in an iterative manner. The best architecture is then taken as the top-1 in the NAO ranking.

3.3. Experimental Setup.

In the next section, we compare the three above-mentioned algorithms with our random search policy, as discussed in Section 3.1.2. To this end, following common practice in NAS, we make use of the word-level language modeling Penn Tree Bank (PTB) dataset (Marcus et al., 1994b). The goal then becomes finding a recurrent cell that correctly predicts the next word given the input sequence. The quality of a candidate cell is then evaluated using the standard *perplexity* metric (Brown et al., 1992).

Note that, for our comparisons, we follow the procedure used in (Liu et al., 2019b; Pham et al., 2018; Luo et al., 2018) for the final evaluation, consisting of keeping the connections found for the best architecture in the search phase but increasing the hidden state size (to 850 in practice), so as to increase capacity. Furthermore, when training an architecture from scratch, we followed Yang et al. (2017); Merity et al. (2017) and make the use of standard SGD first to speed up the training, then change to average SGD to improve the convergence. Note that, for the comparison to be fair, we use the same strategies, for both hidden state size and optimizer, in our random search policy.

4. Experimental Results

We analyze the search phase of the three state-of-the-art NAS algorithms discussed above on the task of language modeling. We first start by comparing these algorithms to our random policy when using the search space of 4 activation functions, which are commonly used in the literature. The surprising findings in this typical NAS use case prompt us to study the behavior of the search strategies in a simplified case, where the search space is reduced to a minimum: 2 nodes. This analysis allows us to identify a factor that has a significant impact on the observed results: Weight sharing. We then quantify this impact on the ranking of the NAS candidates, evidencing that it dramatically affects the effectiveness of the search.

4.1. NAS Comparison in a Standard Search Space

We make use of the three NAS search strategies, DARTS, NAO and ENAS, on the PTB dataset to find a recurrent cell for language modeling, and compare them to our random search policy. For the sake of computational cost, as the experiments are repeated multiple times, we define a search space consisting of 8 nodes, instead of the usual 12 ones, which still yields around *2 billion solutions* ($\mathsf{n}! * |\mathcal{O}|^n$ solutions, where $\mathsf{n} = 8$ nodes and $|\mathcal{O}| = 4$ operations).

For each of the four search policies, we run 10 experiments with a different seed. During the search phase, we used an individual search phase configuration file for each policy, as provided by the respective authors. Once a best architecture is identified by the search phase, it is used for evaluation, i.e., we train the chosen architecture from scratch for 1000 epochs. In this phase, we used a fixed seed for all the tests.

Results. In Figure 3, we plot, on the right, the mean perplexity evolution, over the 1000 epochs, obtained by averaging the results of the best architectures found using the 10 different seeds. On the left of the figure, we show the perplexity evolution for the best cell of each strategy among the 10 different runs. The top figures correspond to DARTS, the middle ones to NAO and the bottom ones to ENAS. In each figure, we also show the curve obtained for the random policy. Specifically, we plot the perplexity of the given policy in blue and that of the random search in orange. The final results of these curves, after 1000 epochs, are further summarized in Table 2.

Altogether, these results show a surprising fact: The overall best cell was found using our random policy. Moreover, on average, only one of the three policies obtains a better

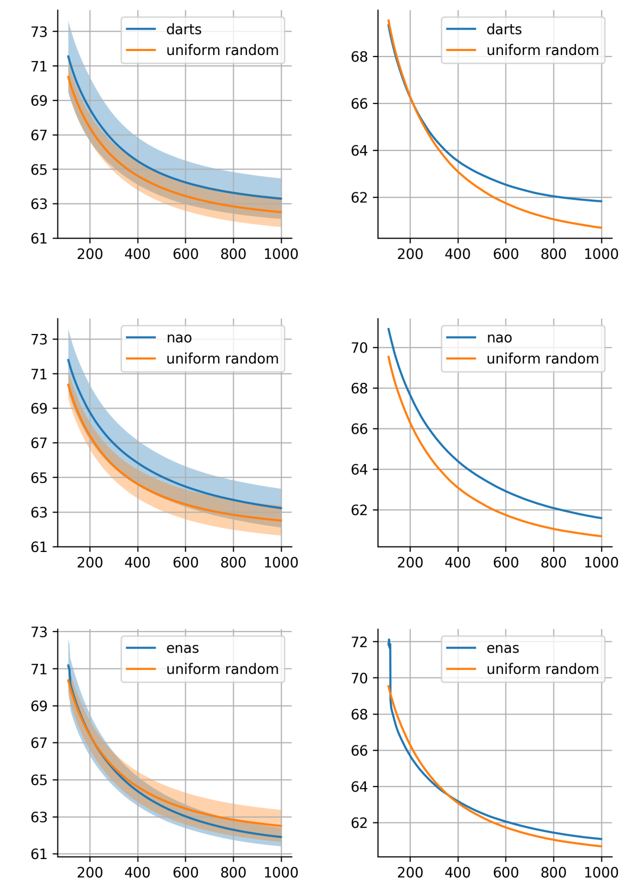

Figure 3. **Perplexity evolution in the 8-node search space. Left:** Mean validation perplexity evolution, for 1000 epochs, averaged over 10 different seeds. Each seed led to a best architecture found using DARTS (top), NAO (middle) and ENAS (bottom). We compare these curves to those of our random policy. **Right:** Results of the best architectures among the 10 different seeds.

TYPE	MEAN VAL	MEAN TEST	BEST VAL	BEST TEST
DARTS	63.29 ± 1.18	60.70 ± 1.12	61.82	59.38
NAO	63.22 ± 1.12	60.79 ± 1.10	61.59	59.21
ENAS	$\mathbf{61.91} \pm \mathbf{0.50}$	$\mathbf{59.25} \pm \mathbf{0.47}$	61.10	58.95
RANDOM	62.50 ± 0.86	60.01 ± 0.86	**60.69**	**58.13**

Table 2. **Final perplexities in the 8-node search space.** We report the mean and best perplexity on the validation and test sets at the end of training the architectures found using DARTS, NAO, ENAS, as well as for our random policy.

performance, but the differences are small. The full learning curves for both the average and best cases in Figure 3 further evidence that this is not an accident; all the policies converge as expected. However, the best results are *not* obtained by the NAS policies.

Observations:

- Random sampling is robust and consistently competitive. As shown in Table 2, it outperforms on average the DARTS and NAO policies, and yields the overall best cell for these experiments.
- The excellent performance of the random policy evidences the high expressiveness of the manually constructed search space; even arbitrary policies in this space perform well, as evidenced by the relatively low standard deviation over the 10 seeds of the random architectures, shown in Fig. 3(left). Existing NAS strategies, including the three policies tested here, typically rely on these spaces, albeit using 12 nodes instead of 8. The influence of such manually built spaces has never been analyzed before.
- Thanks to our framework, we were able for the first time to have a proper comparison among policy strategies. For example, we observed that the ENAS policy sampler has the lowest variance among the three tested ones. This shows that ENAS is more robust to the variance caused by the random seed of the search phase.

Encouraged by these surprising results, we then dig deeper into their causes. To this end, below, we make use of a smaller search space that we can explore exhaustively.

4.2. Searching a Reduced Space

The results in the previous section highlight the inability of the studied methods to surpass the random search. To further understand the reason behind this, we repeat the analysis on the smallest possible space of a similar nature, consisting of only two nodes. Given that each node is identified by two values, the ID of the incoming node and the activation function, the space has a cardinality $|\mathsf{S}| = \mathsf{n}! * |\mathsf{O}|^n$, where $\mathsf{n} = 2$ nodes and $|\mathsf{O}| = 4$ operations, thus only having 32 possible solutions.

As before, we benchmark the same three policies – DARTS, NAO and ENAS – against random sampling. Adapting each policy to the reduced space requires the following changes:

- For DARTS, no changes are needed except directly modifying the number of nodes in the search space.
- For NAO, to mimic the behavior of the algorithm in the space of 8 nodes, we randomly sample 20% of the possible architectures to define the initial candidate pool. We train the encoder-predictor-decoder network for 250 iterations every 50 epochs using the top-4 architectures in the NAO ranking. At each search iteration, we sample at most 3 new architectures to be added to the pool. The rest of the search logic remains unchanged.
- For ENAS, we reduce the number of sampled archi-

TYPE	MEAN VALID	MEAN TEST	BEST VALID	BEST TEST
DARTS	71.29 ± 2.45	68.74 ± 2.42	68.05	65.55
NAO	**68.66 ± 2.50**	**66.03 ± 2.40**	66.22	63.59
ENAS	69.99 ± 0.0	66.61 ± 0.0	69.99	66.61
RANDOM	69.69 ± 2.44	67.21 ± 2.52	**66.16**	**63.27**
REAL-BEST	65.67 ± 0.21	62.95 ± 0.18	65.38	62.63

Table 3. **Final perplexities in the 2-node search space.** We report the mean and best perplexity on the validation and test sets at the end of training the architectures found using DARTS, NAO, ENAS, as well as for our random policy. We also report the real best results obtained by exhaustive search.

tectures in one epoch from 200 to 20 and increase the number of batches to 10 for each sampled architecture. All other hyper-parameters remain the same.

Results. In Table 3, we provide the results of searching the 2-node space. Its smaller size allows us to exhaustively compute the validation and test results of all possible solutions, thus determining the upper bound for this case.

Observations:

- All the policies failed to find the architecture that actually performs best.
- The best architecture sampled by any search policy was found by the random search, as in the 8-node case.
- Even in a space this small, on average, the random policy still yields comparable results to the other search strategies.
- The ENAS policy always converged to the same architecture, as shown by the variance of 0:0 in the table. This further evidences the robustness of ENAS to the random seed.
- NAO performs better than random sampling on average because it has access to a ranking of the architectures. Nevertheless, the final architecture chosen by NAO is *always* one of the architectures from the initial pool, which were sampled uniformly randomly. This indicates that the ranking of NAO is not updated in an appropriate manner throughout the search.

Having observed the incorrect NAO ranking motivated us to further study these architecture rankings, and in particular to connect them to weight sharing. This is what we discuss in the next section.

4.3. Impact of Weight Sharing

As observed above, none of the three NAS methods and random search, each starting the search randomly ten times, managed to find the best architecture out of only 32 possible ones. Not even once. For the random search, this is probable. For the NAS methods, however, it is unexpected. In fact, it suggests that the best architecture only achieves poorly during the search phase of these methods. If the architectures during the search were trained freely, as in the original NAS technique (Zoph & Le, 2017), this would not be possible. Therefore, the only potential culprit for this unexpected behavior is weight sharing (WS); the weights of the best architecture are biased by their use in other candidates. To evidence this, we therefore investigate whether WS reduces the correlation between the ranking in the search phase and the ranking of the evaluation phase.

To this end, we perform the following experiments:

- **Without WS:** We train all the 32 possible architectures of the search space individually. Each architecture is trained 10 times with a different seed, which therefore yields a mean and standard deviation of its performance. The mean value is used as a ground truth – the actual potential of the given architecture.

- **With WS:** We train the 32 architectures in parallel, using the weight sharing strategy employed in NAO and ENAS. As DARTS does not have discrete representations of the solutions during the search, the idea of solution ranking does not apply at that point. During training, each mini-batch is given to an architecture uniformly sampled from the search space. We repeat the process 10 times, with 10 random seeds.

We then compute the correlation between the architecture rankings found with WS and the ground truth (i.e., the architectures trained independently). As a correlation measure, we make use of the *Kendall Tau ( )* metric (Kendall, 1938): a number in the range [-1, 1] with the following properties:

- $= -1$: Maximum disagreement. One ranking is the opposite of the other.
- $= 1$: Maximum agreement. The two rankings are identical.
- close to 0: A value close to zero indicates the absence of correlation.

For each of the 10 runs of the weight sharing strategy, we evaluate the Kendall Tau metric of the final rankings with respect to the real averaged ranking.

Results. In Figure 4, we depict the best, worst and average rankings, where the best and worst ones were found using the Kendall Tau metric. For conciseness, we only show the 10 best-scoring architectures. The color and number in each position indicate the absolute position change for an architecture between the real ranking and the WS one.

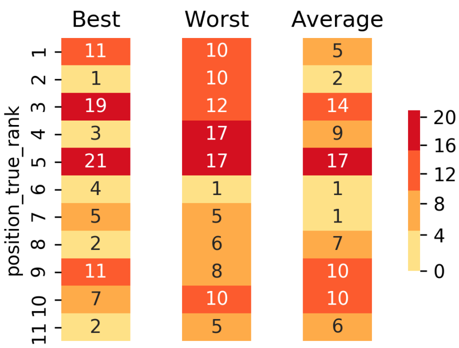

Figure 4. **Ranking changes due to WS.** We show, from left to right, the best, worst and average cases over 10 runs with different random seeds. A change in ranking, indicated by the colors and numbers, is measured as the absolute position change between the WS ranking and the true one. For example, in the Best scenario, the architecture at the top-1 in the true ranking has lost 11 positions in the WS ranking.

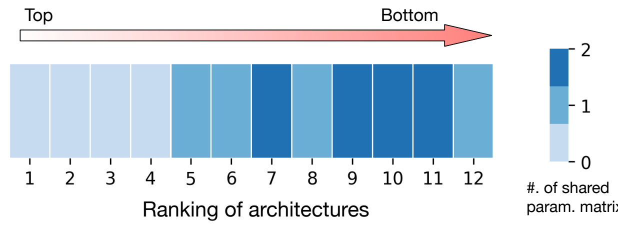

Figure 5. **Influence of the amount of sharing.** We show the ranking obtained using WS, but when fixing 2 out of the 3 nodes in our reduced space. Depending on the active node connections, different architectures share different amounts of weights. The ranking indicates that sharing more weights leads to worse performance.

Observations:

- WS *never* produces the true ranking, as evidenced by the Best case in Figure 4.
- The behavior of the WS rankings is greatly affected by the changing of the seed. In particular, the Kendall Tau values for the three plots in Figure 4 are $0{:}282; -0{:}004; -0{:}116$ for Best, Average and Worst.
- For all of the 10 runs, the Kendall Tau metrics are close to zero, which suggests a lack of correlation between the WS rankings and the true one.
- In the Worst scenario, the first architecture in the WS ranking is at the 14^{th} place in the real ranking, with a validation perplexity of 69.38 at epoch 1000, 4 perplexity points above the real best architecture.

Note that these results were obtained by using the evaluation hyper-parameters to train all models, including the ones with WS. Nevertheless, we observed the same behavior when using the standard search-phase hyper-parameters, which confirms that our findings are not due to a poor hyper-parameter choice.

4.3.1. Influence of the Amount of Sharing

Depending on the active connections in the DAG of Figure 2(a), different architectures are subject to different amounts of weight sharing. For example, let us consider the 3-node case, with node 1 and node 2 fixed and node 2 having node 1 as incoming node. In this scenario, the input to node 3 can be either directly node 0 (i.e., the input), or node 1, or node 2. In the first case, the only network parameters that the output of node 3 depends on are the weights of its own operation, W_3. In the second and third cases, however, the output further depends on the parameters of node 1, W_1, and of nodes 1 and 2, W_1 and W_2, respectively.

To study the influence of the amount of sharing on the architecture ranking, we performed an experiment where we fixed the first two nodes and only searched for the third one. This represents a space of 12 architectures (3 possible connections to node 3 $\times$ 4 operations). The ranking of the 12 architectures is shown in Figure 5, where color indicates the number of shared weight matrices, that is, matrices of nodes 1 and 2, W_1 and W_2, also used in the search for node 3. Note that the top-performing architectures do not share any weights, i.e., only rely on W_3, and that the more weights are shared, the worse the architecture performs.

Together with the previous observations, we believe that these results evidence the negative impact of WS; it dramatically affects the performance of the sampled architectures, thus complicating the overall search process and leading to search policies that are no better than the random one.

5. Conclusion

In this paper, we have provided a thorough analysis of the effectiveness of the search phase of NAS algorithms, including proper and fair comparisons to random search. We have observed that, surprisingly, the search policies of state-of-the-art NAS techniques are no better than random, and have traced the reason for this to the use of (i) a constrained search space and (ii) weight sharing, which shuffles the ranking of architectures during the search phase, and negatively impacts the search.

In essence, our gained insights highlight two key properties of state-of-the-art NAS strategies, which had been overlooked in the past due to the single-minded focus of NAS evaluation on the results on the target task. While we see this as an achievement in itself, we believe that it will also be key to the development of novel search policies for NAS. This will be the focus of our future research.

References

Ahmed, K. and Torresani, L. Maskconnect: Connectivity learning by gradient descent. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 349–365, 2018.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pp. 549–558, 2018.

Brown, P. F., Pietra, V. J. D., Mercer, R. L., Pietra, S. A. D., and Lai, J. C. An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 18(1): 31–40, 1992.

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Efficient architecture search by network transformation. *AAAI*, 2018.

Chen, L.-C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H., and Shlens, J. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, pp. 8713–8724, 2018.

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

Kendall, M. G. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). 2009.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018a.

Liu, C., Chen, L.-C., Schroff, F., Adam, H., Hua, W., Yuille, A., and Fei-Fei, L. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *arXiv preprint arXiv:1901.02985*, 2019a.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. *International Conference on Learning Representations (ICLR), Conference track*, 2018b.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *ICLR*, 2019b.

Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., and Banzhaf, W. Nsga-net: A multi-objective genetic algorithm for neural architecture search. *arXiv preprint arXiv:1810.03522*, 2018.

Luo, R., Tian, F., Qin, T., Chen, E.-H., and Liu, T.-Y. Neural architecture optimization. In *Advances in neural information processing systems*, 2018.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pp. 114–119, Stroudsburg, PA, USA, 1994a. Association for Computational Linguistics. ISBN 1-55860-357-3. doi: 10.3115/1075812.1075835. URL https://doi.org/10.3115/1075812.1075835.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, pp. 114–119. Association for Computational Linguistics, 1994b.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pp. 293–312. Elsevier, 2019.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Pérez-Rúa, J.-M., Baccouche, M., and Pateux, S. Efficient progressive neural architecture search. *arXiv preprint arXiv:1808.00391*, 2018.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *ICML*, 2018.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q., and Kurakin, A. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

Tan, M., Chen, B., Pang, R., Vasudevan, V., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

Xie, L. and Yuille, A. Genetic cnn. *IEEE International Conference on Computer Vision (ICCV)*, 2017.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017.

Zela, A., Klein, A., Falkner, S., and Hutter, F. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *ICML AutoML Workshop*, 2018.

Zoph, B. and Le, Q. V. Neural Architecture Search with Reinforcement Learning. *International Conference on Learning Representations (ICLR), Conference track*, 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.