A APPENDIX

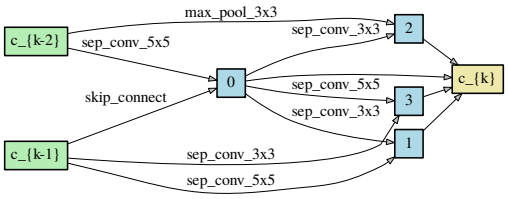
A.1 BACKGROUND

We first provide an overview of the components of hyperparameter optimization and, by association, NAS. As shown in Figure 1, a general hyperparameter optimization problem has three components, each of which can have NAS-specific approaches. We provide a brief overview of the components below, drawing attention to NAS-specific methods (see the survey by Elsken et al. [12] for a more thorough coverage of NAS).

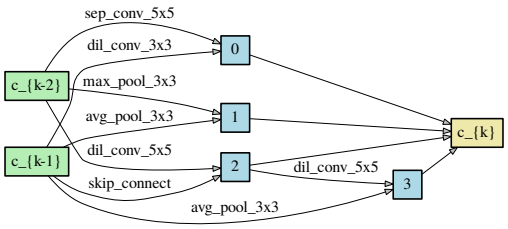
Search Space. Hyperparameter optimization involves identifying a good hyperparameter configuration from a set of possible configurations. The search space defines this set of configurations, and can include continuous or discrete hyperparameters in a structured or unstructured fashion [42, 3, 14, 38]. NAS-specific search spaces usually involve discrete hyperparameters with additional structure that can be captured with a directed acyclic graph (DAG) [39, 32]. Additionally, since a search space for designing an entire architecture would have too many nodes and edges, search spaces are usually defined over some smaller building block, i.e., cell blocks, that are repeated in some way via a preset or learned meta-architecture to form a larger architecture [12]. We design our random search NAS algorithm for such a cell block search space, using the same search spaces for the CIFAR-10 and PTB benchmarks as DARTS for our experiments in Section 4. See Section 3 for a concrete example of one such search space.

Search Method. Given a search space, there are various search methods to select putative configurations to evaluate. Random search is the most basic approach, yet it is quite effective in practice [3, 27]. Various general and NAS-specific adaptive methods have also been introduced, all of which attempt to bias the search in some way towards configurations that are more likely to perform well. In traditional hyperparameter optimization, the choice of search method can depends on the search space. Bayesian approaches based on Gaussian processes [42, 25, 43, 22] and gradient-based approaches [2, 34] are generally only applicable to continuous search spaces. In contrast, tree-based Bayesian [18, 4], evolutionary strategies [38], and random search are more flexible and can be applied to any search space. NAS-specific search methods can also be categorized into the same broad categories but are tailored for structured NAS search spaces (see Section 2.2 for a more involved discussion).

Evaluation Method. For each hyperparameter configuration considered by a search method, we must evaluate its quality. The default approach to perform such an evaluation involves fully training a model with the given hyperparameters, and subsequently measuring its quality, e.g., its predictive accuracy on a validation set. The first generation of NAS methods relied on full training evaluation, and thus required thousands of GPU days to achieve a desired result [49, 40, 50, 41]. In contrast, partial training methods exploit early-stopping to speed up the evaluation process at the cost of noisy estimates of configuration quality. These methods use Bayesian optimization [25, 22, 13], performance prediction [15, 10], or multi-armed bandits [20, 27, 28] to adaptively allocate resources to different configurations. NAS-specific evaluation methods exploit the structure of neural networks to provide even cheaper, heuristic estimates of quality. Many of these methods center around sharing and reuse: network morphisms build upon previously trained architectures [6, 11, 21]; hypernetworks and performance prediction encode information from previously seen architectures [5, 29, 48]; and weight-sharing methods [39, 32, 1, 45, 7] use a single set of weights for all possible architectures.



(a) Normal Cell



(b) Reduction Cell

Figure 3: **Convolutional Cells on CIFAR-10 Benchmark:** Best architecture found by random search with weight-sharing.

A.2 REPRODUCIBILITY OF RECENT WORK

We summarize the reproducibility of recent NAS publications at some of the major machine learning conferences in Table 5 according to the availability of the following:

1. **Architecture search code.** The output of this code is the final architecture that should be trained on the evaluation task.

Table 5: **Reproducibility of NAS Publications.** Summary of the reproducibility status of recent NAS publications appearing in top machine learning conferences. For the hyperparameter tuning column, N/A indicates we are not aware that the authors performed additional hyperparameter optimization.
† Published result is not reproducible for the PTB benchmark when training the reported final architecture with provided code.
‡ The authors stated they will release the entire codebase upon publication.
* Code to reproduce experiments was requested on OpenReview.

Conference	Publication	Architecture Search Code	Model Evaluation Code	Random Seeds	Hyperparameter Tuning
ICLR 2018	Brock et al. [5]	Yes	Yes	No	N/A
	Liu et al. [30]	No	No		
ICML 2018	Pham et al. [39]†	Yes	Yes	No	Undocumented
	Cai et al. [6]	Yes	Yes	No	N/A
	Bender et al. [1]	No	No		
NIPS 2018	Kandasamy et al. [24]	Yes	Yes	No	N/A
	Luo et al. [33]	Yes	Yes	No	Grid Search
ICLR 2019	Liu et al. [32]	Yes	Yes	No	Undocumented
	Cai et al. [7]‡	No	Yes	No	N/A
	Zhang et al. [48]*	No	No		
	Xie et al. [45]*	No	No		
	Cao et al. [8]	No	No		

2. **Model evaluation code.** The output of this code is the final performance on the evaluation task.

3. **Hyperparameter tuning documentation.** This includes code used to perform hyperparameter tuning of the final architectures, if any.

4. **Random Seeds.** This includes random seeds used for both the search and post-processing (i.e., retraining of final architecture as well as any additional hyperparameter tuning) phases. Most works provide the final architectures but random seeds are required to verify that the search process actually results in those final architectures and the performance of the final architectures matches the published result. Note the random seeds are only useful if the code for search and post-processing phases are deterministic up to a random seed; this was not the case for the DARTS code used for the CIFAR-10 benchmark.

All 4 criteria are necessary for exact reproducibility. Due to the absence of random seeds for all methods with released code, none of the methods in Table 5 are exactly reproducible from the search phase to the final architecture evaluation phase. Additionally, while only criteria 1–3 are necessary to estimate broad reproducibility, there is minimal discussion of the broad reproducibility of existing methods in published work.

A.3 PTB BENCHMARK

We now present results for the PTB benchmark. We use the DARTS search space for the recurrent cell, which is described in Section 3. For this benchmark, due to higher memory requirements for their mixture operation, DARTS used a small recurrent network with embedding and hidden dimension of 300 to perform the architecture search followed by a larger network with embedding and hidden dimension of 850 to perform the evaluation. For the PTB benchmark, we refer to the network used in the first stage as the *proxy* network and the network in the later stages as the *proxyless* network.

Our setup matches that of DARTS with the following exceptions:

Architecture Operations. In stage (1), DARTS trained the shared weights network with the zero operation included in the list of considered operations but removed the zero operation when selecting the final architecture to evaluate in stages (2) and (3). For our random search with weight-sharing, we decided to exclude the zero operation for both search and evaluation.

Stage 3 Procedure. For stage (3) evaluation, we follow the ArXiv version of DARTS [31], which reported two sets of results, one after training for 1600 epochs and another fine tuned result after training for an additional 1000 epochs. In the ICLR version, Liu et al. [32] simply say they trained the final network to convergence. We trained for another 1000 epochs for a total of 3600 epochs to approximate training to convergence.

We next present the final search results. We subsequently explore the impact of various meta-hyperparameters on random search with weight-sharing, and finally evaluate the reproducibility of various methods on this benchmark.

A.3.1 Final Search Results

We now present our final evaluation results in Table 6. Specifically, we report the output of stage (3), in which we train the proxyless network configured according to the best architectures found by different methods for 3600 epochs. We discuss various aspects of these results in the context of the three issues—baselines, complex methods, reproducibility—introduced in Section 1.

First, we evaluate the ASHA baseline using 2 GPU days, which is equivalent to the total cost of DARTS (second order). In contrast to the one random architecture evaluated by Pham et al. [39] and the 8 evaluated by Liu et al. [32] for their random search baselines, ASHA evaluated over 300 architectures with the allotted computation time. The best architecture found by ASHA achieves a test perplexity of 56.4, which is comparable to the published result for ENAS and significantly better than the random search baseline provided by Liu et al. [32], DARTS (first order), and the reproduced result for ENAS [32]. Our result demonstrates that the gap between SOTA NAS methods and standard hyperparameter optimization approaches on the PTB benchmark is significantly smaller than that suggested by the existing comparisons to random search [39, 32].

Next, we evaluate random search with weight-sharing with tuned meta-hyperparameters (see Section A.3.2 for details). With slightly lower search cost than DARTS, this method finds an architecture that reaches test perplexity 55.5, achieving SOTA perplexity compared to previous NAS approaches. We note that manually designed architectures are competitive with RNN cells designed by NAS methods on this benchmark. In fact, the work by Yang et al. [47] using LSTM with mixture of experts in the softmax layer (MoS) outperforms automatically designed cells. Our architecture would likely also improve significantly with MoS, but we train without MoS to provide a fair comparison to ENAS and DARTS.

Finally, we examine the reproducibility of the NAS methods with available code for both architecture search and evaluation. For DARTS, exact reproducibility was not feasible since Liu et al. [32] do not provide random seeds for the search process; however, we were able to reproduce the performance of their reported best architecture. We also evaluated the broad reproducibility of DARTS through an independent run, which reached a test perplexity of 55.9, compared to the published value of 55.7. For ENAS, end-to-end exact reproducibility was infeasible due to non-deterministic code and missing random seeds for both the search and evaluation steps. Additionally, when we tried to reproduce their result using the provided final architecture, we could not match the reported test perplexity of 56.3 in our rerun. Consequently, in Table 6 we show the test perplexity for the final architecture found by ENAS trained using the DARTS code base, which Liu et al. [32] observed to give a better test perplexity than using the architecture evaluation code provided by ENAS. We next considered the reproducibility of random search with weight-sharing. We verified the exact reproducibility of our reported results, and then investigated their broad reproducibility by running another experiment with different random seeds. In this second experiment, we observed a final text perplexity of 56.5, compared with a final test perplexity of 55.5 in the first experiment. Our detailed investigation in Section A.3.3 shows that the discrepancies across both DARTS and random search with weight-sharing are unsurprising in light of the differing convergence rates among architectures on this benchmark.

A.3.2 Impact of Meta-Hyperparameters

We now detail the meta-hyperparameter settings that we tried for random search with weight-sharing in order to achieve SOTA performance on the PTB benchmark. Similar to DARTS, in these preliminary experiments we performed 4 separate trials of each version of random search with weight-sharing, where each trial consists of executing stage (1) followed by stage (2). In stage (1), we train the shared weights and then use them to evaluate 2000 randomly sampled architectures. In stage (2), we select the best architecture out of 2000, according to the shared weights, to train from scratch using the proxyless network for 300 epochs.

We incrementally tune random search with weight-sharing by adjusting the following meta-hyperparameters associated with training the shared weights in stage (1): (1) gradient clipping, (2) batch size, and (3) network size. The settings we consider proceed as follows:

Random (1): We train the shared weights of the proxy network using the same setup as DARTS with the same values for number of epochs, batch size, and gradient clipping; all other meta-hyperparameters are the same.

Random (2): We decrease the maximum gradient norm to account for discrete architectures, as opposed to the weighted combination used by DARTS, so that gradient updates are not as large in each direction.

Random (3): We decrease batch size from 256 to 64 in order to increase the number of architectures used to train the shared weights.

Random (4): We train the larger proxyless network architecture with shared weights instead of the proxy network, thereby significantly increasing the number of parameters in the model.

The stage (2) performance of the final architecture after

Table 6: **PTB Benchmark: Comparison with state-of-the-art NAS methods and manually designed networks.** Lower test perplexity is better on this benchmark. The results are grouped by those for manually designed networks, published NAS methods, and the methods that we evaluated. Table entries denoted by "-" indicate that the field does not apply, while entries denoted by "N/A" indicate unknown entries. The search cost, unless otherwise noted, is measured in GPU days. Note that the search cost is hardware dependent and the search cost shown for our results are calculated for Tesla P100 GPUs; all other numbers are those reported by Liu et al. [32].
[#] Search cost is in CPU-days.
[*] We could not reproduce this result using the code released by the authors at https://github.com/melodyguan/enas.
[†] The stage (1) cost shown is that for 1 trial as opposed to the cost for 4 trials shown for DARTS and Random search WS. It is unclear whether ENAS requires multiple trials followed by stage (2) evaluation in order to find a good architecture.

		Test Perplexity		Params	Search Cost			Comparable	Search
Architecture	**Source**	Valid	Test	(M)	Stage 1	Stage 2	Total	**Search Space?**	**Method**
LSTM + DropConnect	[36]	60.0	57.3	24	-	-	-	-	manual
ASHA + LSTM + DropConnect	[28]	58.1	56.3	24	-	-	13	N	HP-tuned
LSTM + MoS	[47]	56.5	54.4	22	-	-	-	-	manual
NAS[#]	[49]	N/A	64.0	25	-	-	1e4	N	RL
ENAS[*†]	[39]	N/A	56.3	24	0.5	N/A	N/A	Y	RL
ENAS[†]	[32]	60.8	58.6	24	0.5	N/A	N/A	Y	random
Random search baseline	[32]	61.8	59.4	23	-	-	2	Y	random
DARTS (first order)	[32]	60.2	57.6	23	0.5	1	1.5	Y	gradient-based
DARTS (second order)	[32]	58.1	55.7	23	1	1	2	Y	gradient-based
DARTS (second order)	Ours	58.2	55.9	23	1	1	2	Y	gradient-based
ASHA baseline	Ours	58.6	56.4	23	-	-	2	Y	random
Random search WS	Ours	57.8	55.5	23	0.25	1	1.25	Y	random

retraining from scratch for each of these settings is shown in Table 7. With the extra capacity in the larger network used in Random (4), random search with weight-sharing achieves average validation perplexity of 64.7 across 4 trials, with the best architecture (shown in Figure 2 in Section 3) reaching 63.8. In light of these stage (2) results, we focused in stage (3) on the best architecture found by Random (4) Run 1, and achieved test perplexity of 55.5 after training for 3600 epochs as reported in Table 6.

A.3.3 Investigating Reproducibility

We next examine the stage (2) intermediate results in Table 7 in the context of reproducibility. The first two rows of Table 7 show a comparison of the published stage (2) results for DARTS and our independent runs of DARTS. Both the best and average across 4 trials are worse in our reproduction of their results. Additionally, as previously mentioned, we perform an additional run of Random (4) with 4 different random seeds to test the broad reproducibility our result. The minimum stage (2) validation perplexity over these 4 trials is 63.9, compared to a minimum validation perplexity of 63.8 for the first set of seeds.

Next, in Table 8 we compare the validation perplexities of the best architectures from ASHA, Random (4) Run 1, Random (4) Run 2, and our independent run of DARTS after training each from scratch for up to 3600 epochs. The swap in relative ranking across epochs demonstrates the risk of using noisy signals for the reward. In this case, we see that even partial training for 300 epochs does not recover the correct ranking; training using shared weights further obscures the signal. The differing convergence rates explain the difference in final test perplexity of the best architecture from Random (4) Run 2 and those from DARTS and Random (4) Run 1, despite Random (4) Run 2 reaching a comparable perplexity after 300 epochs.

Overall, the results of Tables 7 and 8 demonstrate a high variance in the stage (2) intermediate results across trials, along with issues related to differing convergence rates for different architectures. These two issues help explain the differences between the independent runs of DARTS and random search with weight-sharing. A third potential source of variation, which could in particular adversely impact our random search with weight-sharing results, stems from the fact that we did not perform any additional hyperparameter tuning in stage (3); instead we used the same training hyperparameters that were tuned by Liu et al. [32] for the final architecture found by DARTS.

A.4 CIFAR-10 BENCHMARK

In this section, we provide additional detail for the experiments in Section 4.1.

Architecture Operations. In stage (1), DARTS trained the shared weights network with the zero operation included in the list of considered operations but removed

Table 7: **PTB Benchmark: Comparison of Stage (2) Intermediate Search Results for Weight-Sharing Methods.** In stage (1), random search is run with different settings to train the shared weights. The resulting networks are used to evaluate 2000 randomly sampled architectures. In stage (2), the best of these architectures for each trial is then trained from scratch for 300 epochs. We report the performance of the best architecture after stage (2) across 4 trials for each search method.

	Setting				Trial					
Method	Network Config	Epochs	Batch Size	Gradient Clipping	1	2	3	4	Best	Average
DARTS [32]	proxy	50	256	0.25	67.3	66.3	63.4	63.4	**63.4**	**65.1**
Reproduced DARTS	proxy	50	256	0.25	64.5	67.7	64.0	67.7	64.0	66.0
Random (1)	proxy	50	256	0.25	65.6	66.3	66.0	65.6	65.6	65.9
Random (2)	proxy	50	256	0.1	65.8	67.7	65.3	64.9	64.9	65.9
Random (3)	proxy	50	64	0.1	66.1	65.0	64.9	64.5	64.5	65.1
Random (4) Run 1	proxyless	50	64	0.1	66.3	64.6	64.1	63.8	**63.8**	**64.7**
Random (4) Run 2	proxyless	50	64	0.1	63.9	64.8	66.3	66.7	63.9	65.4

Table 8: **PTB Benchmark: Ranking of Intermediate Validation Perplexity.** Architectures are retrained from scratch using the proxyless network and the validation perplexity is reported after training for the indicated number of epochs. The final test perplexity after training for 3600 epochs is also shown for reference.

	Validation Perplexity by Epoch										Test Perplexity	
	300		500		1600		2600		3600			
Search Method	Value	Rank	Value	Rank	Value	Rank	Value	Rank	Value	Rank	Value	Rank
DARTS	64.0	4	61.9	2	59.5	2	58.5	2	58.2	2	55.9	2
ASHA	63.9	2	62.0	3	59.8	4	59.0	3	58.6	3	56.4	3
Random (4) Run 1	63.8	1	61.7	1	59.3	1	58.4	1	57.8	1	55.5	1
Random (4) Run 2	63.9	2	62.1	4	59.6	3	59.0	3	58.8	4	56.5	4

the zero operation when selecting the final architecture to evaluate in stages (2) and (3). For our random search with weight-sharing, we decided to *include* the zero operation for both search and evaluation. We hypothesize that our results may improve if we impose a higher complexity on the final architectures by excluding the zero operation.

Stage 1 Procedure. For random search with weight-sharing, after the shared weights are fully trained, we evaluate randomly sampled architectures using the shared weights and select the best one for stage (2) evaluation. Due to the higher cost of evaluating on the full validation set, we evaluate each architecture using 10 minibatches instead. We split the total number of architectures to be evaluated into sets of 1000. For each 1000, we select the best 10 according the cheap evaluation on part of the validation set and evaluate on the full validation set. Then we select the top architecture across all sets of 1000 for stage (2) evaluation.

Reproducibility. The code released by Liu et al. [32] did not produce deterministic results for the CNN benchmark due to non-determinism in CuDNN and in data loading. We removed the non-deterministic behavior in CuDNN by setting

```
cudnn.benchmark = False
cudnn.deterministic = True
cudnn.enabled=True
```

Note that this only disables the non-deterministic functions in CuDNN and does not adversely affect training time as much as turning off CuDNN completely. We fix additional non-determinism from data loading by setting the seed for the `random` package in addition to `numpy.random` and `pytorch` seed and turning off multiple threads for data loading.

We ran ASHA and one set of trials for random search with weight-sharing using the non-deterministic code before fixing the seeding to get deterministic results. Hence, the result for ASHA does not satisfy exact reproduciblity due to non-deterministic training and asynchronous updates. Due to the demanding computational cost of these experiments, we use the non-deterministic runs of random with weight-sharing as the second set of trials for Random (5) in Table 2, all other settings for random search with weight-sharing are deterministic.

A.5 AVAILABLE CODE

Unless otherwise noted, our results are exactly reproducible from architecture search to final evaluation using the code available at `https://github.com/liamcli/randomNAS_release`. The code we use for random search with weight-sharing on both benchmarks is deterministic conditioned on a fixed random seed. We provide the final architectures used for each of

the trials shown in the tables above, as well as the random seeds used to find those architectures. In addition, we perform no additional hyperparameter tuning for final architectures and only tune the meta-hyperparameters according to the discussion in the text itself. We also provide code, final architectures, and random seeds used for our experiments using ASHA. However, we note that there is one uncontrolled source of randomness in our ASHA experiments—in the distributed setting, the asynchronous nature of the algorithm means that the results depend on the order in which different architectures finish (partially) training. Lastly, our experiments were conducted using Tesla P100 and V100 GPUs on Google Cloud. We convert GPU time on V100 to equivalent time on P100 by applying a multiple of 1.5.