# DARTS without a Validation Set: Optimizing the Marginal Likelihood

**Miroslav Fil**[1*] **Binxin Ru**[1] **Clare Lyle**[1] **Yarin Gal**[1]

[1] OATML Group, Department of Computer Science, University of Oxford, UK

## Abstract

The success of neural architecture search (NAS) has historically been limited by excessive compute requirements. While modern weight-sharing NAS methods such as DARTS are able to finish the search in single-digit GPU days, extracting the final best architecture from the shared weights is notoriously unreliable. Training-Speed-Estimate (TSE), a recently developed generalization estimator with a Bayesian marginal likelihood interpretation, has previously been used in place of the validation loss for gradient-based optimization in DARTS. This prevents the DARTS skip connection collapse, which significantly improves performance on NASBench-201 and the original DARTS search space. We extend those results by applying various DARTS diagnostics and show several unusual behaviors arising from not using a validation set. Furthermore, our experiments yield concrete examples of the depth gap and topology selection in DARTS having a strongly negative impact on the search performance despite generally receiving limited attention in the literature compared to the operations selection.

## 1 Introduction

Neural architecture search (NAS) algorithms have been able to automatically find network architectures that outperform the best human designs in test set performance on several benchmarks, making it an important subfield of AutoML [10, 23]. However, earliest specialized NAS methods using evolutionary algorithms [22] or reinforcement learning [32] required thousands of GPU days to achieve good performance because each architecture was trained separately. Weight-sharing algorithms such as ENAS [21] or DARTS [18], in which all the architectures share the same set of weights in a large supernetwork, have reduced the computational cost to single-digit GPU days while often delivering better performance. However, identifying the single best architecture based on the shared weights is known to be unstable [2, 18, 30]. DARTS in particular suffers from overfitting to too many skip connections [28], and Dong et al. [7] have shown that DARTS might even select architectures with all skip connections and no parameter-based operations.

Most weight-sharing NAS extracts the best architecture out of the supernetwork either by evaluating the validation accuracy of single architectures with weights inherited from the supernetwork [17, 13], or by explicitly optimizing the validation loss end-to-end in a differentiable fashion as in DARTS [18]. However, the performance of architectures using the shared weights tends to be significantly different from their performances when trained separately [2, 30], which makes it difficult to identify the standalone best architecture.

Ru et al. [24] have shown that Training-Speed-Estimate (TSE), a recent generalization estimator based on Bayesian model selection [20] that does not require a validation set, can be used to provide

*Correspondence to miroslav.fil@cs.ox.ac.uk.

5th Workshop on Meta-Learning at NeurIPS 2021, held virtually.

high fidelity yet computationally cheap early-stopping estimates of architecture test set performance in non-weight-sharing NAS. They also showed promising results on TSE being applicable to various weight-sharing NAS algorithms. Our work extends those results by providing additional benchmarks of TSE applied to DARTS (TSE-DARTS) and its variants on the NASBench-series. The main novelty of our work lies in showing that using TSE both improves performance and fundamentally changes the behavior of DARTS, thus expanding the insights from previous work on explaining DARTS [28, 25].

## 2 Background

We first introduce the basics behind DARTS and TSE, and then follow up by showing how to efficiently compute the gradients of TSE to use for the differentiable optimization in DARTS.

### 2.1 DARTS

DARTS [18] searches for a single architecture cell that is stacked repeatedly to form the final network. Such search space design is ubiquitous in modern NAS work [21, 22, 17]. A cell in the DARTS search space is simply a directed acyclic graph (DAG) with 7 nodes. The DAG is fully connected, and every edge between a pair of nodes $(i, j)$ represents one operation $o$ from the 8 operations in the search space (such as skip connections, zero op, convolution or max pool). The output $x^j$ of intermediate nodes in the DAG is equal to a sum of the operations corresponding to all the edges $(\cdot, j)$ with input $x^i$:

$$x^j = \sum_{i<j} o^{(i,j)}(x^i) \tag{1}$$

During the search, DARTS relaxes the constraint that each edge $(i, j)$ must represent only a single operation, and instead uses a softmax with learnable weights $\alpha$ to make the sum differentiable. Therefore, the node output is a mixture of all the available operations at once defined as

$$\overline{o}^{(i,j)} = \sum_{o \in O} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})} o(x). \tag{2}$$

DARTS searches for normal and reduction cells. Most of the network is composed of normal cells, whereas there are two reduction cells in total at $1/3$ and $2/3$ of the network depth. The difference is that reduction cells have all operations with stride two. The whole architecture encoding can thus be expressed as $\alpha = (\alpha_{normal}, \alpha_{reduce})$. DARTS formulates the optimization of the weights $w$ and architecture $\alpha$ as a bi-level optimization problem

$$\min_{\alpha} \quad L_{val}(w^*(\alpha), \alpha) \tag{3a}$$

$$\text{s. t.} \quad w^*(\alpha) = \text{argmin}_w L_{train}(w, \alpha) \tag{3b}$$

$L_{train}$ and $L_{val}$ are losses on the train and validation sets, respectively. This formulation is equivalent to the training loops in other problems such as meta-learning [11] or gradient-based hyperparameter optimization [1]. The optimal weights $w^*$ are approximated by only a single step of SGD, which means that the architecture gradients actually descend $L_{val}(w - \eta \nabla_w L_{train}(w, \alpha))$. At the end of training, the supernetwork is discretized into a single architecture by taking argmax over each edge to select the operation that has the highest edge within the softmax. It is therefore implicitly assumed that the most useful operations will simultaneously have the highest weights. In practice, selecting the architecture this way is known to result in too many skip connections [28, 5], which frequently results in poor performance of the final architecture.

### 2.2 TSE

TSE is defined as the sum over training losses during a model's optimization. Let $D = \{(x_1, y_1), (x_2, y_2), .., (x_n, y_n)\}$ be the training dataset, $f_{\theta_t}(x_{i(t)})$ represent a model's output for the training sample $x_{i(t)}$ in the $t$-th iteration with $\theta_t$ being the model weights at time $t$, $L$ be a loss function and $T$ be the total training iterations. We define TSE as:

$$\text{TSE} = \sum_{t=1}^{T} L(f_{\theta_t}(x_{i(t)}), y_{i(t)}). \tag{4}$$

Lyle et al. [20] have shown that TSE provably corresponds to an evidence lower bound in Bayesian linear regression, making it an estimate of the marginal likelihood. For non-linear models such as neural networks, TSE is a theoretically-inspired metric that was already shown to be useful as a predictor of generalization in NAS by Ru et al. [24]. Similar optimization-based generalization estimators have also been noted to correlate well with generalization by Jiang et al. [15].

It is possible to integrate TSE into DARTS by replacing descent of the validation loss gradient $\nabla L_{val}(w^*(\alpha), \alpha)$ by descending the TSE gradient. Note that optimizing TSE amounts to minimizing the sum over training losses during training. Therefore, when using TSE-DARTS, no validation set is required, and we only use the training set to find both the weights and the architecture. We compute TSE over $T = 100$ SGD iterations and iteratively update the architecture as in normal DARTS. Algorithm 1 shows the TSE-DARTS training loop, which simply uses different gradients for updating the architecture compared to the original DARTS training loop [18]. We discuss how to compute the TSE gradient in Section 2.3.

**Algorithm 1:** TSE-DARTS

Create mixed operations $\overline{o}^{(i,j)}$ parameterized by $\alpha^{(i,j)}$ for each edge $(i, j)$
Set T=100 steps of unrolling for computing TSE
**while** *not converged* **do**
1. Approximate the optimal weights $w^*$ with T steps of SGD by computing $w_T = w_0 - \eta \sum_{t=0}^{T} \nabla_w L_{train}(f(x_t, w_t, \alpha), y_t)$;
2. Update architecture $\alpha$ by descending TSE gradient via Eq. (6);
3. Update the original weights with T steps of SGD $w_T = w_0 - \eta \sum_{t=0}^{T} \nabla_w L_{train}(f(x_t, w_t, \alpha), y_t)$ using the new architecture encoding;

**end**

### 2.3 Computing the TSE gradient

While optimizing TSE instead of the validation loss is conceptually simple, obtaining the TSE gradients is very computationally demanding in practice. Naively computing TSE gradient requires differentiating through the whole training history to compute gradients with respect to the architecture $\alpha$, which leads to $O(T)$ memory requirements due to the workings of reverse-mode auto-differentiation [1]. This would make it impossible to use large-scale networks. Precisely, the exact gradient for the final training loss at time $T$ after training with SGD is equal to (proof is included in Appendix A):

$$\nabla_\alpha L_{train}^T = \frac{\partial L_{train}^T}{\partial \alpha} + \frac{\partial L_{train}^T}{\partial w}(-\eta \sum_{0 \le j \le T} ([\prod_{0 \le k < j} I - \eta \frac{\partial^2 L_{train}^{T-k-1}}{\partial w \partial w}] \frac{\partial^2 L_{train}^{T-j-1}}{\partial w \partial \alpha})), \tag{5}$$

where we abbreviate $L_{train}(f(x_T, w_T, \alpha), y_T)$ as $L_{train}^T$. The TSE gradient would be equivalent to summing such gradients over all the time steps $0, 1, .., T$. Instead, we follow Ru et al. [24], who proposed to use a first-order approximation for each training loss gradient that only includes the direct gradient $\frac{\partial L_{train}^T}{\partial \alpha}$ from Eq. (12) at each time step. The whole TSE gradient can then be approximated as

$$\nabla_\alpha \text{TSE} = \nabla_\alpha (L_{train}^0 + L_{train}^1 + .. + L_{train}^T) \approx \sum_{t=1}^{T} \frac{\partial L_{train}^t}{\partial \alpha} \tag{6}$$

This form of the TSE gradient has several advantages. Most importantly, the $\nabla_\alpha L_{train}^t$ can be computed for free concurrently with the weights gradients. As a result, computing the approximate TSE gradient has the same time and memory costs as normal training via backprop. An example implementation is shown in Appendix D.

## 3 Experiments

We first show the performance of TSE-DARTS compared to baseline DARTS on NASBench-201 [7] and NASBench-1shot1 [27]. While those search spaces are small, they tabulate the ground truth test set performance of every architecture in the search space. The tabular nature of those benchmarks

| | | CIFAR10 |
|---|---|---|
| NASBench-201 | SPOS [17] | 91.05 (0.66) |
| | GDAS [8] | 93.23 (0.58) |
| | ENAS [21] | 93.76 (0.00) |
| | DARTS [18] | 65.38 (7.84) |
| | TSE-DARTS | 92.66 (0.00) |
| NB101-3 | DARTS [18] | 93.35 (0.01) |
| | TSE-DARTS | 87.87 (0.00) |

Table 1: Summary of the performance of TSE-DARTS on NASBench-201 and NASBench-1shot1 search space 3 (abbreviated as NB101-3). TSE-DARTS has a much stronger performance than DARTS on NASBench-201 as it is able to avoid collapsing to all skip connections. However, DARTS does better on NB101-3 because of better topology as discussed in Section 3.3.

makes it especially viable to use those spaces for diagnosing the behavior of TSE-DARTS, which we do in Sections 3.3 and 3.4 as our main empirical contribution. For the DARTS search space, we retrain our best architectures using the 600 epochs training protocol as originally used by DARTS [18]. Moreover, we use the DARTS search protocol, where we first run the search four times, retrain all the resulting architectures once, and then retrain the best out of the four for three more seeds to compute its mean ground truth test set accuracy. We also evaluate TSE versions of DrNAS [4] and PDARTS [5]. Our experiments with TSE algorithms always reuse the default parameters from their corresponding public implementations.

### 3.1 NASBench-201 and NASBench-1shot1

Searching on NASBench-201 [7] is a notorious example of DARTS overfitting to skip connections as the final selected architecture often contains only skip connections with no parameter-based operations. This makes DARTS have a consistently poor performance. Table 1 shows that using TSE-DARTS is able to prevent the skip connection collapse and achieve $92.66\%$ test set accuracy, which is competitive with other weight-sharing algorithms such as GDAS [8] or SPOS [17] in NASBench-201. On the other hand, DARTS has a strong performance on all three search spaces from NASBench-1shot1 [27], from which we show results on search space 3, denoted as NB101-3. Note that none of the search spaces have skip connections in the operations set, hence DARTS cannot overfit to them. In comparison, TSE-DARTS fares poorly on this benchmark. Section 3.3 shows that this is due to picking cells that are too deep rather than due to the operations choice, which is unusual given the consistent overfitting to skip connections in normal DARTS. Appendix B includes results on additional NASBench-201 datasets and NASBench-1shot1 search spaces, which are mostly analogous to those already shown.

### 3.2 DARTS search space and the role of depth

Table 2 shows the results of TSE-DARTS on the original DARTS search space. The best architecture found by TSE-DARTS achieves $2.73\%$ test set accuracy, which is a marginal improvement over the $2.76\%$ of second-order DARTS. However, it is a significant improvement compared to the $3.00\%$ of first-order DARTS, which is particularly notable since both TSE-DARTS and first-order DARTS have the same compute and memory requirements. Moreover, TSE-DARTS finds very different architectures compared to normal DARTS. Even in the DARTS search space, there is no overfitting to skip connections for TSE-DARTS. Instead, the discovered architectures tend to be very heavy on separable convolutions, often with no parameter-free operations at all. The best architecture found by TSE-DARTS is visualized in Appendix C.

We hypothesize that skip connections are never chosen because they are unnecessary in the DARTS search setup, which uses a shallow search supernetwork. In contrast, note that skip connections are known to be more beneficial for deeper architectures [14, 16]. The search uses only 8 layers in the supernetwork even though the evaluation with retraining from scratch is done using 20 layers networks. This inequity between the search and evaluation networks is often referred to as the depth gap. To test whether TSE-DARTS would pick skip connections with a deeper search supernetwork, we also ran the search using 20 layers (denoted as TSE-DARTS-20) instead of the original 8 layers.

| | 8 layers eval (% err. rate) | 20 layers eval (% err. rate) |
|---|---|---|
| DARTS (1st) [18] | 4.87 (0.18) | 3.00 (0.14) |
| DARTS (2nd) [18] | 4.91 (0.22) | 2.76 (0.09) |
| DrNAS [4] | 4.47 (0.28) | 2.46 (0.03) |
| PC-DARTS [26] | 4.31 (0.03) | 2.57 (0.07) |
| PDARTS [5] | 4.47 (0.15) | 2.50 |
| iDARTS [29] | 4.35 (0.09) | 2.37 (0.03) |
| TSE-DARTS | 4.64 (0.22) | 2.73 (0.12) |
| TSE-DrNAS | 4.35 (0.19) | 2.90 (0.14) |
| TSE-PDARTS | 4.61 (0.22) | 3.19 (0.12) |
| TSE-DARTS-20 | 4.47 (0.18) | 2.68 (0.04) |
| DARTS-20 (first-order) | 5.06 (0.39) | 2.89 (0.11) |

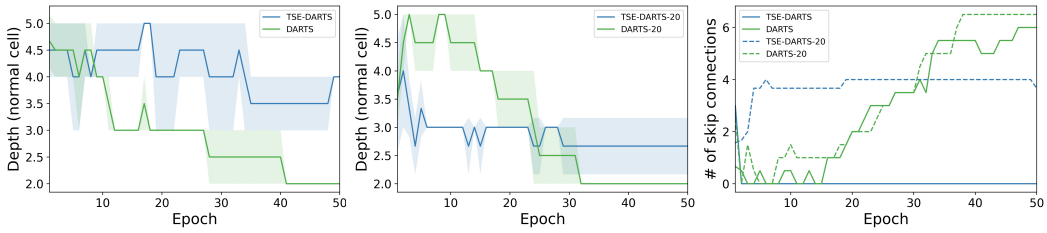
Table 2: Summary of performances of TSE variants against baselines in the DARTS search space. TSE-DARTS outperforms the baseline DARTS, but the results from applying it to other state-of-the-art algorithms are mixed. DARTS (1st) and (2nd) refer to first and second-order DARTS, respectively.

Figure 1c compares the number of skip connections for TSE-DARTS and TSE-DARTS-20. The increased depth clearly favors picking a moderate amount of skip connections without overfitting to them as in original DARTS, whereas the shallow TSE-DARTS has no skip connections whatsoever. The best architecture of TSE-DARTS-20 has standalone test accuracy of $2.68\%$, further improving on the $2.73\%$ of TSE-DARTS. We also benchmarked first-order DARTS-20, which reached $2.89\%$ final test set accuracy, improving on the $3.00\%$ of original first-order DARTS.

To better assess the effects of depth gap, we also trained the standalone architectures from scratch using 8 layers for the evaluation. This again completely eliminates the depth gap except that now both the search and evaluation networks are shallow. Table 2 shows the results with 8 layers evaluation using the same architectures discovered by each search algorithm as the corresponding 20 layers evaluation. In this setup, the TSE variants consistently improve on their baseline versions. This is true particularly for TSE-DrNAS, which has a weak performance of $2.90\%$ in the original 20 layers evaluation setup, but top tier results in 8 layers evaluation. Because some of the TSE variants significantly improve in the 8 layers evaluation but are close or worse than the respective baselines using 20 layers evaluation, it suggests that the architectural optimum with 8 layers networks can be quite different from the 20 layers optimum. Hence, even if the search algorithm itself is improved, it might still lead to worse results in the DARTS 20 layers evaluation if the search only uses 8 layers, since the performance gains might not transfer across different depths due to the depth gap.

Related work including PDARTS [5] and DrNAS [4] tries to tackle the depth gap by progressively deepening the search network while removing some operations from the operations set during multiple phases of the search. We also tried to do this and tested TSE-PDARTS, but we found the resulting architectures to be very poor with only $3.19\%$ accuracy. Furthermore, the final architectures never had any skip connections because the skip connections already got removed from the search space while the network was still shallow. This suggests that even though the progressive deepening might be a key factor in PDARTS's impressive $2.50\%$ performance, the improvement might not necessarily be coming from reducing the depth gap since the architectures found by TSE-DARTS-20 and TSE-PDARTS are very different.

It thus appears that searching without the depth gap can be more effective compared to progressive deepening. DARTS keeps the search supernetwork small primarily because of excessive memory requirements due to the continuous relaxation. However, the full-depth search is computationally feasible when using techniques such as gradient accumulation, which allows to trade-off extra compute time for memory, or partial channel connections as in PC-DARTS [26]. The TSE-DARTS-20 search only takes around 1 day to complete, which is the same as second-order DARTS, and around 23GB of VRAM when using the original 64 batch size. Note that optimizing TSE has the added advantage of making DARTS less memory-intensive due to not using any second-order gradients, hence making it easier to use higher depth supernetworks. At the same time, searching with full depth




(a) Normal cell (8 layers) (b) Normal cell (20 layers) (c) # of skip

Figure 1: When searching on the DARTS search space, baseline DARTS is biased towards shallow architectures. Architectures found by TSE-DARTS tend to be significantly deeper for both 8 and 20 layers search as shown in a) and b), respectively. The range of depth in the DARTS space is [2, 5], hence DARTS attains the minimum depth possible in the normal cell. c) shows the number of skip connections, where we note that TSE-DARTS-20 picks a stable, moderate amount of skip connections while DARTS overfits to them over time.

supernetworks might even be computationally cheaper overall if it were to improve on the search instability in DARTS, reducing the number of seeds that need to be ran to obtain good results.

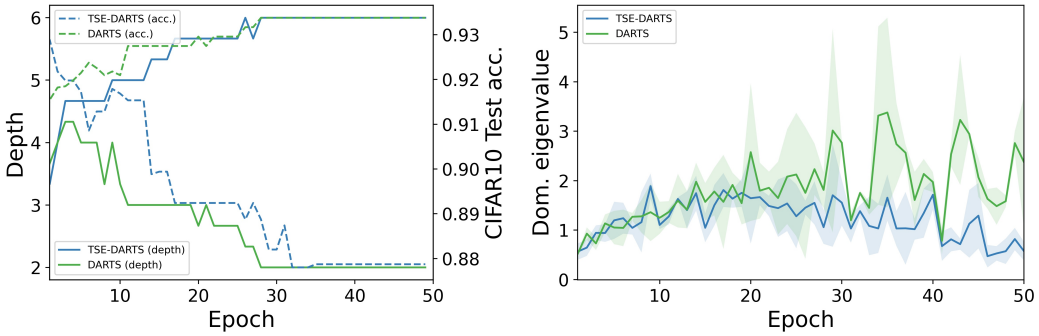
### 3.3 Bias towards shallow architectures

In Section 3.2, we investigated the effect of the number of layers in the supernetwork considered as a fixed hyperparameter. Now, we investigate the depth of the architectural cells themselves which is determined by the search. Shu et al. [25] have shown that DARTS tends to be biased towards shallow cells. This is explained by shallower architectures having smoother loss landscapes, which makes them easier to train. DARTS favors them since the gradient-based search is naturally greedy towards architectures which train fast one step ahead.

Following Shu et al. [25], we measure depth as the length of the longest path from input to output within the cell DAG. Figure 1 shows that TSE-DARTS tends to select significantly deeper cells than normal DARTS. This is inconsistent with the intuition that DARTS is biased towards shallow architectures because they train faster, since optimizing TSE means that the bias towards architectures which achieve low training loss faster should be even more prominent. We propose an explanation for this paradox in Section 3.4. Additionally, we found the depth of TSE-DARTS-20 architectures to be inbetween DARTS and TSE-DARTS, whereas DARTS-20 again found the shallowest cells, same as normal DARTS. It appears that TSE-DARTS with both 8 and 20 layers maintains a roughly constant total depth of the network since searching with 8 cells finds deep individual cells whereas searching with 20 cells finds shallower cells.

Next, we consider the selection of skip connections, which have been shown to make the loss landscapes of deep neural networks smoother as deep networks tend to otherwise have very non-smooth loss regions [16]. Therefore, TSE-DARTS-20 picking more skip connections can also be interpreted in the context of smoothing the loss landscapes, which only becomes necessary when the search supernetwork has 20 layers. Skip connections might have limited utility with only 8 layers because the network is still shallow, and hence are not selected by TSE-DARTS. Our results would thus suggest that the bias towards shallow architectures observed by Shu et al. [25] is more strongly related to the higher loss landscape smoothness of such architectures rather than skip connections making the architecture train faster, which is often believed to be the reason for DARTS's bias towards skip connections [31, 5]. Otherwise, TSE-DARTS would have even stronger overfitting to skip connections than normal DARTS, which is the opposite of what happens empirically.

We further show that the poor results of TSE-DARTS on NASBench-1shot1 are caused by finding architectures with excessive depth. Both DARTS and TSE-DARTS find architectures which are predominantly if not all 3x3 convolutions, but the performance of DARTS is high because it finds the shallowest cells possible. Note that NASBench-1shot1 search spaces contain no skip connections, which makes it impossible for DARTS to overfit to them. Figure 2a shows the performance of the architectures selected by DARTS and TSE-DARTS on NB101-3 alongside the cell depth, where lower depth clearly correlates with better performance.




(a) NB101-3 - test acc. and depth

(b) NB101-3 - dom. eigenvalue

Figure 2: a) TSE-DARTS only gets worse over time on NB101-3 whereas baseline DARTS achieves top tier performance. Note the strong negative correlation between depth and final performance because both algorithms select $3 \times 3$ convolutions for most if not all operations during the search. In b), we expand the results of Zela et al. [28] as discussed in Section 3.4, and show that TSE-DARTS has decreasing eigenvalues along with decreasing performance at the same time.

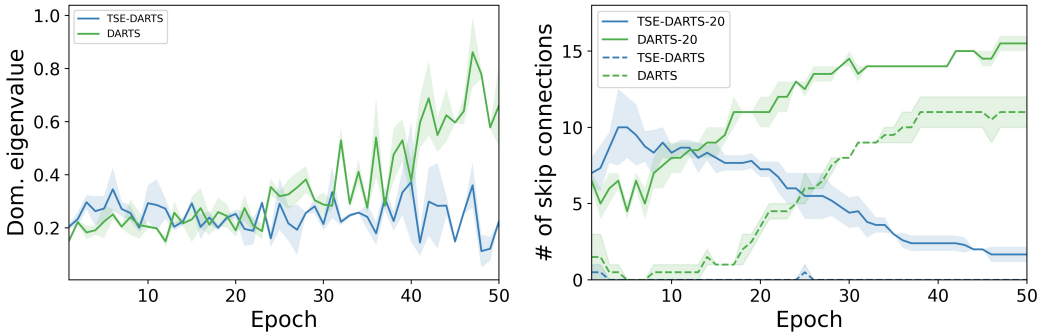
Overall, DARTS consistently finds shallow architectures even without any skip connections in the search space while TSE-DARTS finds deep cells. Whether this leads to good or bad performance then depends on the search space. The results here also highlight that selecting the cell topology can be a serious issue in differentiable NAS as the poor performance of TSE-DARTS on NASBench-1shot1 is almost entirely caused by the suboptimal cell topology. However, most work on improving DARTS focuses on the skip connection collapse rather than improving the topology selection.

### 3.4 Eigenvalues of the architecture Hessian

Zela et al. [28] argue that the skip connection overfitting in DARTS is related to the growing dominant eigenvalues of the architecture Hessian. We additionally confirm the trend of rising eigenvalues for DARTS on the NASBench search spaces [6, 27], which have been released after the original work by Zela et al. [28]. On the other hand, we also show that the behavior of TSE-DARTS eigenvalues is much more varied.

We first reproduce experiments on the reduced S2 search space proposed by Zela et al. [28], which contains only skip connections and $3 \times 3$ separable convolutions for two operations total. Figure 3 shows that TSE-DARTS picks zero skip connections even on this space whereas DARTS severely overfits to skip connections. DARTS-20 overfits to skip connections even more while TSE-DARTS-20 chooses a small but nonzero number of skip connections, which attests to skip connections being more prominent when the search supernetwork is deep. The architecture eigenvalues increase for DARTS but stay at low levels for TSE-DARTS. Figure 2 shows the same experiment on NB101-3, where we see that the eigenvalues for TSE-DARTS actually go down but so does its performance, whereas DARTS has rising eigenvalues and rising performance. On NASBench-201, the eigenvalues increase for both DARTS and TSE-DARTS. However, the performance keeps improving for TSE-DARTS but it gets worse over time for DARTS. We visualize the NASBench-201 results in Appendix E.

Overall, it appears that the eigenvalue phenomenon observed by Zela et al. [28] also has significant differences for DARTS and TSE-DARTS. While DARTS always has rising eigenvalues and simultaneously overfits to skip connections, neither appears to be the case for TSE-DARTS. A potential explanation might be that TSE-DARTS is more biased towards architectures with smooth loss landscapes as proposed by Shu et al. [25], which by definition results in smaller architecture Hessian eigenvalues by the virtue of the search algorithm finding architectures without excessively sharp loss regions. This would be in disagreement with the observation in Section 3.3 showing that TSE-DARTS discovers deeper architectures, which should instead have less smooth loss landscapes. Nonetheless, we see that in practice, DARTS has higher eigenvalues despite finding shallower architectures. A plausible explanation might be that the architecture depth is confounded by the architecture selection in DARTS being suboptimal as we had explicitly shown in Section 3.3. The TSE-DARTS bias towards deep architectures might then be caused by the architecture selection procedure rather than




(a) S2 space - dom. eigenvalue

(b) S2 space - # of skip

Figure 3: a) The architecture eigenvalues of TSE-DARTS on the S2 search space do not increase unlike in DARTS. b) shows that the increased search supernetwork depth further exacerbates the overfitting to skip connections for DARTS-20, which picks up to 16 out of 16 operations as skip connections. TSE-DARTS-20 is able to avoid it, and picks a moderate amount of skip connections in the final architecture while shallow TSE-DARTS chooses zero skip connections at all times.

the optimization itself. An explicit treatment of cell topology could therefore allow to better reconcile our results with those of Shu et al. [25] and Zela et al. [28]. That said, note that analyzing the supernetwork eigenvalues already has the advantage of being independent of architecture selection, and TSE-DARTS having smaller architecture eigenvalues can also be understood as supporting evidence for TSE-DARTS to be finding shallow architectures.

## 4 Conclusion

In this work, we analyzed TSE-DARTS, a variant of DARTS where the architecture is updated to minimize TSE rather than validation loss. Our results show that TSE-DARTS tends to have a stronger performance than DARTS, and we demonstrated that TSE-DARTS also has several interesting qualitative behaviors. In particular, it is free of the skip connection collapse that is known to be a major issue in DARTS. We applied several DARTS diagnostics previously proposed in the literature and shown that TSE-DARTS has fundamentally different biases than the original DARTS. TSE-DARTS appears to be finding significantly deeper architectures than DARTS [25], and its architecture Hessian eigenvalues appear to have different trajectories depending on the search space, whereas the DARTS eigenvalues rise consistently as observed by Zela et al. [28]. Our experiments also yielded a concrete example of the depth gap that cannot be fixed with progressive deepening [5, 4], where TSE-DARTS with an increased number of layers in the search supernetwork starts selecting skip connections despite not picking any otherwise. We have also shown that the DARTS architecture selection via argmax on the architecture coefficients can lead to problems with the cell topology rather than just the selected operations as is commonly assumed.

Ultimately, while TSE-DARTS is able to outperform the baseline DARTS, it appears to have its own distinct set of problems, and it cannot be trivially applied to other state-of-the-art DARTS variants to further advance their performance. However, the follow-up work on DARTS might be effectively overfitted to fixing DARTS's problems, which are no longer present in TSE-DARTS. It is possible that if the same amount of attention was devoted to fixing the problems of TSE-DARTS, it could achieve much better performance. Such extensions are an interesting direction for future work.

## Acknowledgments and Disclosure of Funding

Binxin Ru was supported by the Clarendon Scholarship. Clare Lyle was supported by the Open Philantropy Foundation AI Fellows Program.

## References

[1] Atilim Gunes Baydin et al. "Automatic Differentiation in Machine Learning: a Survey". In: *Journal of Machine Learning Research* 18.153 (2018), pp. 1–43. ISSN: 1533-7928.

[2] Gabriel Bender et al. "Understanding and simplifying one-shot architecture search". In: *International Conference on Machine Learning*. PMLR, 2018, pp. 550–559.

[3] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13.10 (2012), pp. 281–305. ISSN: 1533-7928.

[4] Xiangning Chen et al. "Dr{NAS}: Dirichlet Neural Architecture Search". In: *International Conference on Learning Representations*. 2021.

[5] Xin Chen et al. "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1294–1303.

[6] Xuanyi Dong and Yi Yang. "NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search". In: *arXiv:2001.00326 [cs]* (Jan. 15, 2020). arXiv: 2001.00326.

[7] Xuanyi Dong and Yi Yang. "Nas-bench-201: Extending the scope of reproducible neural architecture search". In: *arXiv preprint arXiv:2001.00326* (2020).

[8] Xuanyi Dong and Yi Yang. "Searching for A Robust Neural Architecture in Four GPU Hours". In: *arXiv:1910.04465 [cs]* (Oct. 16, 2019). arXiv: 1910.04465.

[9] Xuanyi Dong et al. "NATS-Bench: Benchmarking nas algorithms for architecture topology and size". In: *IEEE transactions on pattern analysis and machine intelligence* (2021). Publisher: IEEE.

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural architecture search: A survey". In: *The Journal of Machine Learning Research* 20.1 (2019). Publisher: JMLR. org, pp. 1997–2017.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 17, 2017, pp. 1126–1135.

[12] Luca Franceschi et al. "Forward and Reverse Gradient-Based Hyperparameter Optimization". In: *International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 17, 2017, pp. 1165–1173.

[13] Zichao Guo et al. "Single path one-shot neural architecture search with uniform sampling". In: *European Conference on Computer Vision*. Springer, 2020, pp. 544–560.

[14] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 1063-6919. June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[15] Yiding Jiang et al. "Fantastic Generalization Measures and Where to Find Them". In: Eighth International Conference on Learning Representations. Apr. 2020.

[16] Hao Li et al. "Visualizing the Loss Landscape of Neural Nets". In: *Advances in Neural Information Processing Systems* 31 (2018).

[17] Liam Li and Ameet Talwalkar. "Random search and reproducibility for neural architecture search". In: *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 367–377.

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "DARTS: Differentiable Architecture Search". In: *International Conference on Learning Representations*. 2018.

[19] Jonathan Lorraine, Paul Vicol, and David Duvenaud. "Optimizing Millions of Hyperparameters by Implicit Differentiation". In: *International Conference on Artificial Intelligence and Statistics*. International Conference on Artificial Intelligence and Statistics. ISSN: 2640-3498. PMLR, June 3, 2020, pp. 1540–1552.

[20] Clare Lyle et al. "A Bayesian Perspective on Training Speed and Model Selection". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 10396–10408.

[21] Hieu Pham et al. "Efficient Neural Architecture Search via Parameter Sharing". In: *arXiv:1802.03268 [cs, stat]* (Feb. 11, 2018). arXiv: 1802.03268.

[22] Esteban Real et al. "Regularized evolution for image classifier architecture search". In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33. Issue: 01. 2019, pp. 4780–4789.

[23] Pengzhen Ren et al. "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions". In: *ACM Computing Surveys* 54.4 (May 22, 2021), 76:1–76:34. ISSN: 0360-0300. DOI: 10.1145/3447582.

[24] Binxin Ru et al. "Speedy Performance Estimation for Neural Architecture Search". In: *arXiv:2006.04492 [cs, stat]* (June 7, 2021). arXiv: 2006.04492.

[25] Yao Shu, Wei Wang, and Shaofeng Cai. "Understanding Architectures Learnt by Cell-based Neural Architecture Search". In: Eighth International Conference on Learning Representations. Apr. 2020.

[26] Yuhui Xu et al. "PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search". In: *International Conference on Learning Representations*. 2019.

[27] Arber Zela, Julien Siems, and Frank Hutter. "NAS-Bench-1Shot1: Benchmarking and Dissecting One-shot Neural Architecture Search". In: Eighth International Conference on Learning Representations. Apr. 2020.

[28] Arber Zela et al. "Understanding and Robustifying Differentiable Architecture Search". In: *International Conference on Learning Representations*. 2019.

[29] Miao Zhang et al. "iDARTS: Differentiable Architecture Search with Stochastic Implicit Gradients". In: *arXiv:2106.10784 [cs]* (June 20, 2021). arXiv: 2106.10784.

[30] Yuge Zhang, Quanlu Zhang, and Yaming Yang. "How Does Supernet Help in Neural Architecture Search?" In: *arXiv:2010.08219 [cs]* (May 5, 2021). arXiv: 2010.08219.

[31] Pan Zhou et al. "Theory-Inspired Path-Regularized Differential Network Architecture Search". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 8296–8307.

[32] Barret Zoph et al. "Learning transferable architectures for scalable image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.

## Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See the Conclusions section.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our work has no direct societal impact.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Appendix A, which proves our only theoretical result
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A, which proves our only theoretical result.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Most hyperparameters are defaults from the corresponding benchmarks. We specified the parameters specific to our method.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All tabular results include error bars and so do the figures where clarity allows.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We specified the search and evaluation costs of our methods as well as the baselines.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cite all the NASBench and other relevant baseline code implementations of methods that we adapt. We used no other relevant data or models apart from well-known public datasets such as CIFAR10.

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable?

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

# A Derivation of the exact TSE gradient

The derivation here is analogous to those in other work involving differentiation through optimization [19, 12]. Assume we optimize the weights by SGD, then the weights $w_T$ after $T$ time steps from initial $w_0$ can be written as

$$w_T = w_0 - \eta \sum_{t=0}^{T} \nabla_w L_{train}(f(x_t, w_t, \alpha), y_t) \tag{7}$$

With this in mind, we will now compute the final training loss gradient after T iterations $\nabla_\alpha L_{train}(f(x_T, w_T, \alpha), y_T)$ as the first step to computing the TSE gradients, where we abbreviate $L_{train}(f(x_T, w_T, \alpha), y_T)$ as $L^T_{train}$:

$$\nabla_\alpha L^T_{train} = \frac{\partial L^T_{train}}{\partial \alpha} + \frac{\partial L^T_{train}}{\partial w} \frac{\partial w_T}{\partial \alpha} \tag{8}$$

The $\frac{\partial L^T_{train}}{\partial \alpha}$ is sometimes referred to as the direct gradient, and the $\frac{\partial L^T_{train}}{\partial w} \frac{\partial w_T}{\partial \alpha}$ is also known as the indirect gradient or hypergradient. Now given the expression for $w_T$ given in Eq. (7), we calculate

$$\begin{aligned} \frac{\partial w_T}{\partial \alpha} &= \frac{\partial}{\partial \alpha}(w_{T-1} - \eta \frac{\partial L^{T-1}_{train}}{\partial w}) \\ &= \frac{\partial w_{T-1}}{\partial \alpha} - \eta(\frac{\partial^2 L^{T-1}_{train}}{\partial w \partial \alpha} \frac{\partial \alpha}{\partial \alpha} + \frac{\partial^2 L^{T-1}_{train}}{\partial w \partial w} \frac{\partial w_{T-1}}{\partial \alpha}) \\ &= -\eta \frac{\partial^2 L^{T-1}_{train}}{\partial w \partial \alpha} + (I - \eta \frac{\partial^2 L^{T-1}_{train}}{\partial w \partial w}) \frac{\partial w_{T-1}}{\partial \alpha} \end{aligned} \tag{9}$$

The $\frac{\partial w_{T-1}}{\partial \alpha}$ at the end of Eq. (9) gives a recurrent relation that can be further expanded to give

$$\begin{aligned}\frac{\partial w_T}{\partial \alpha} &= -\eta \frac{\partial^2 L_{train}^{T-1}}{\partial w \partial \alpha} + (I - \eta \frac{\partial^2 L_{train}^{T-1}}{\partial w \partial w})(-\eta \frac{\partial^2 L_{train}^{T-2}}{\partial w \partial \alpha} + (I - \eta \frac{\partial^2 L_{train}^{T-2}}{\partial w \partial w}) \frac{\partial w_{T-2}}{\partial \alpha}) \\ &= -\eta \frac{\partial^2 L_{train}^{T-1}}{\partial w \partial \alpha} - \eta (I - \eta \frac{\partial^2 L_{train}^{T-1}}{\partial w \partial w}) \frac{\partial^2 L_{train}^{T-2}}{\partial w \partial \alpha} + \prod_{0 \leq k < 2} [I - \eta \frac{\partial^2 L_{train}^{T-k-1}}{\partial w \partial w}] \frac{\partial w_{T-2}}{\partial \alpha}\end{aligned} \tag{10}$$

If we unroll the whole history until $w_0$, for which it holds $\frac{\partial w_0}{\partial \alpha} = \mathbf{0}$, we get a string of summands that can be summarized as

$$\frac{\partial w_T}{\partial \alpha} = -\eta \sum_{0 \leq j \leq T} ([\prod_{0 \leq k < j} I - \eta \frac{\partial^2 L_{train}^{T-k-1}}{\partial w \partial w}] \frac{\partial^2 L_{train}^{T-j-1}}{\partial w \partial \alpha}) \tag{11}$$

In total, we have that Eq. (8) is equivalent to

$$\nabla_\alpha L_{train}^T = \frac{\partial L_{train}^T}{\partial \alpha} + \frac{\partial L_{train}^T}{\partial w} (-\eta \sum_{0 \leq j \leq T} ([\prod_{0 \leq k < j} I - \eta \frac{\partial^2 L_{train}^{T-k-1}}{\partial w \partial w}] \frac{\partial^2 L_{train}^{T-j-1}}{\partial w \partial \alpha})) \tag{12}$$

This is the exact unrolled differentiation hypergradient using the last training loss. The TSE gradient computed over T steps of training is simply equal to a sum of the individual training loss gradients:

$$\nabla_\alpha TSE = \sum_{t=0}^{T} (\frac{\partial L_{train}^t}{\partial \alpha} + \frac{\partial L_{train}^t}{\partial w} (-\eta \sum_{0 \leq j \leq t} ([\prod_{0 \leq k < j} I - \eta \frac{\partial^2 L_{train}^{t-k-1}}{\partial w \partial w}] \frac{\partial^2 L_{train}^{t-j-1}}{\partial w \partial \alpha}))) \tag{13}$$

## B Additional results on NASBench-201 and NASBench-1shot1

We also show the results of TSE-DARTS on all the three datasets in NASBench-201 [6] and all the three search spaces in NASBench-1shot1 [27]. For NASBench-201, we additionaly show that training for 150 rather than 50 epochs still retains strong performance and the robustness to skip connection collapse holds even over longer training.

| | CIFAR10 | CIFAR100 | ImageNet16-120 |
|---|---|---|---|
| REA [22] | 94.02 (0.31) | 72.23 (0.95) | 45.77 (0.80) |
| Random Search [3] | 93.90 (0.26) | 71.86 (0.89) | 45.28 (0.97) |
| SPOS [17] | 91.05 (0.66) | 68.26 (0.96) | 40.69 (0.36) |
| GDAS [8] | 93.23 (0.58) | 68.17 (2.50) | 39.40 (0.00) |
| ENAS [21] | 93.76 (0.00) | 70.67 (0.62) | 41.44 (0.00) |
| DrNAS [4] | 93.76 (0.00) | 68.82 (2.06) | 41.44 (0.00) |
| DARTS (first-order) [18] | 59.84 (7.84) | 61.26 (4.43) | 37.88 (2.91) |
| DARTS (second-order) [18] | 65.38 (7.84) | 60.49 (4.95) | 36.79 (7.59) |
| TSE-DARTS | 92.66 (0.00) | 68.13 (2.18) | 36.60 (0.00) |
| TSE-DARTS (150 epochs) | 89.89 (2.52) | 71.03 (0.66) | 33.75 (0.00) |
| TSE-DrNAS | 93.76 (0.00) | 71.11 (0.00) | 41.44 (0.00) |

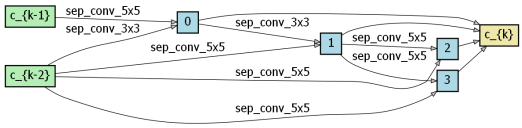
Table 3: Baseline DARTS overfits to all skip connections on NASBench-201, which leads to very poor performance. TSE-DARTS prevents this and achieves $92.66\%$ test set accuracy. Most baselines were reprinted from Dong et al. [9] except the DrNAS results, which are our own. Notably, we were unable to reproduce the original results from Chen et al. [4] with near-perfect performance using the public code.

| | NB101-1 | NB101-2 | NB101-3 |
|---|---|---|---|
| DARTS | 93.33 (0.01) | 93.37 (0.00) | 93.35 (0.01) |
| TSE-DARTS | 91.86 (0.68) | 90.15 (0.00) | 87.87 (0.00) |

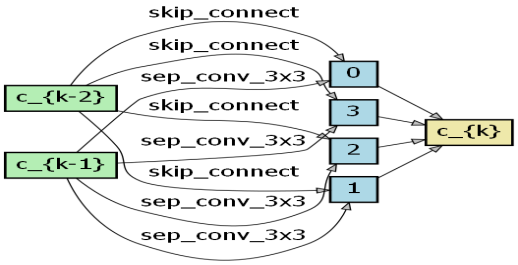
Table 4: On NASBench-1shot1, the performance of TSE-DARTS is quite weak overall and significantly trails baseline DARTS on all the three search spaces. NB101-1, NB101-2 and NB101-3 abbreviate the NASBench-1shot1 search spaces 1, 2 and 3, respectively.

## C Best discovered architectures by TSE-DARTS

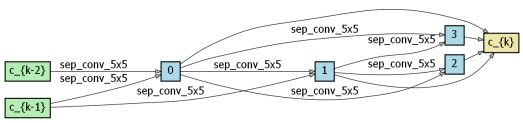
Figure 4 shows the final best architectures found by TSE-DARTS and TSE-DARTS-20. Notably, TSE-DARTS finds significantly deeper architectures for both the normal and reduction cells than TSE-DARTS-20.



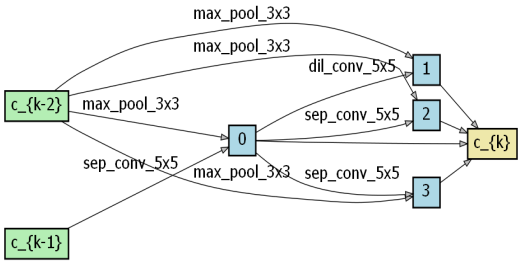

(a) TSE-DARTS-8 - normal cell




(b) TSE-DARTS-20 - normal cell




(c) TSE-DARTS-8 - reduction cell




(d) TSE-DARTS-20 - reduction cell

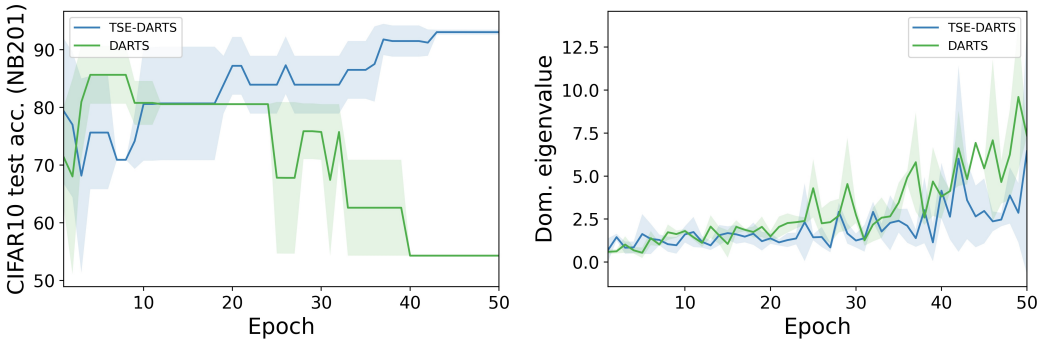
Figure 4: Overview of the best normal and reduction cells found by TSE-DARTS-8 and TSE-DARTS-20 on the DARTS search space.

## D TSE-DARTS implementation

We showcase an example PyTorch implementation of Algorithm 1 (TSE-DARTS) to demonstrate that TSE is very easy to implement on top of standard training loops. The most important part is on lines 12-17, which show that it is trivial to accumulate the approximate TSE gradient by not zeroing out the architecture gradients after each weight update. All implementations of TSE-DARTS variants proceed analogously to this example.

```
1  def train_TSE(train_queue, network, criterion, w_optimizer, a_optimizer, T=100):
2      train_iter = iter(train_queue)
3      network.train()
4
5      for unrolling_step in range(math.ceil(len(train_queue)/T)):
6        # format_input_data outputs a list of T (input, output) pairs
7        all_base_inputs, all_base_targets = format_input_data(train_iter, T=T)
8        network.zero_grad()
9        model_init = deepcopy(network.state_dict()) # Save weights before unrolling so they can be restored later
10
11       # Step 1 of Algorithm 3 - do the unrolling over 100 steps to collect TSE gradient
12       for (base_inputs, base_targets) in zip(all_base_inputs, all_base_targets):
13           logits = network(base_inputs)
14           base_loss = criterion(logits, base_targets)
15           base_loss.backward()
16           w_optimizer.step() # Train the weights during unrolling as normal,
17           w_optimizer.zero_grad() # but the architecture gradients are not zeroed during the unrolling
18
19
20       # Step 2 of Algorithm 3 - update the architecture encoding using accumulated gradients
21       a_optimizer.step()
22
23       a_optimizer.zero_grad() # Reset to get ready for new unrolling
24       w_optimizer.zero_grad()
25
26       new_arch_params = deepcopy(network.arch_params) # Temporary backup for new architecture encoding
27
28       network.load_state_dict(model_init) # Old weights are loaded, which also reverts the architecture encoding
29       for p1, p2 in zip (network.arch_params, new_arch_params):
30         p1.data = p2.data
31
32       # Step 3 of Algorithm 3 - training weights after updating the architecture encoding
33       for (base_inputs, base_targets) in zip(all_base_inputs, all_base_targets):
34         logits = network(base_inputs)
35         base_loss = criterion(logits, base_targets)
36         base_loss.backward()
37         w_optimizer.step()
38
39         w_optimizer.zero_grad()
40         a_optimizer.zero_grad()
```

## E Eigenvalues on NASBench-201




(a) NB201 - test acc. on CIFAR10

(b) NB201 - dom. eigenvalue

Figure 5: On NASBench-201, both DARTS and TSE-DARTS have rising eigenvalues. However, the DARTS performance keeps going down while TSE-DARTS performance increases consistently.