LIGHT AND ACCURATE: NEURAL ARCHITECTURE SEARCH VIA TWO CONSTANT SHARED WEIGHTS INITIALISATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

In recent years, zero-cost proxies are gaining grounds in the field of neural architecture search (NAS). These methods allow to find the optimal neural network for a given task faster and with lesser computational load than conventional NAS methods. Equally important is the fact that they also shed some light on the internal workings of neural architectures. In this paper we present a zero-cost metric that is highly correlated with the train accuracy across the NAS-Bench-101, NAS-Bench-201 and NAS-Bench-NLP benchmark datasets. Architectures are initialised with two distinct constant shared weights, one at a time. Then, a fixed random mini-batch of data is passed forward through each initialisation. We observe that the dispersion of the outputs between two initialisations is positively correlated with trained accuracy. The correlation further improves when the dispersion is normalised by the average output magnitude. Our metric does not require gradients computation and true labels. It thus unbinds NAS procedure from training hyperparameters, loss metric and human-labelled data. Our method is easy to integrate within existing NAS algorithms and takes a fraction of second to evaluate a single network.

1 INTRODUCTION

The field of neural architecture search (NAS) has emerged about a decade ago as an effort to automatise the process of neural geometry optimisation. At the early stages of NAS field development, every candidate architecture was evaluated through training process (reinforcement learning (Williams, 1992), evolutionary algorithms (Real et al., 2019), Bayesian optimisation (Falkner et al., 2018; White et al., 2021)).

One-shot algorithms adopting weight sharing alleviate the necessity of multiple architectures training, reducing the search time drastically (efficient reinforcement learning (Pham et al., 2018), random search with parameters sharing(Li & Talwalkar, 2020), differentiable methods (Liu et al., 2018; Dong & Yang, 2019a;b; Chu et al., 2020)). Yet, they require the training of a massive hypernet, which necessitates elaborate hyperparameters tuning. While these methods prove efficient, they do not always achieve satisfactory results (Dong & Yang, 2019b). One of the best performing of them, DARTS- (Chu et al., 2020), shows significant uncertainty comparing to evolutionary or reinforcement algorithms.

There also exist methods that estimate network performance without training the dataset of interest, but relying on an auxiliary predictive machine learning (ML) model built on a dataset of trained architectures (Istrate et al., 2019; Deng et al., 2017). These methods aim to accelerate the NAS process for image recognition, but still rely on training and cannot be applied to other ML problems.

Evaluating geometries through training clearly brings multiple disadvantages. The most obvious of them is that training is a computationally expensive process, and large-scale geometry evaluation often cannot be carried on massive datasets. As consequence, architectures are usually trained with a single random seed and a fixed set of hyperparameters. This raises the question as to whether a chosen architecture is statistically reliable, and might lead to selecting a sub-optimal model, optimal only in the context of the chosen set of hyperparameters. Training also implies usage of hand-labelled data, which brings in human error – ImageNet dataset, for instance, is known to have a label error of

about $6\,\%$ (Northcutt et al., 2021). Importantly, from the fundamental point of view, the above NAS methods do not provide an explanation as to why a given architecture is chosen.

1.1 ZERO-COST NAS

To alleviate the process of architecture search, many researchers are focused on developing methods that allow to find optimal architectures without model training – so-called zero-cost NAS methods. Here, networks are evaluated via some proxy metric, which typically requires the equivalent of one or a few training epochs. These methods are therefore 2-3 orders of magnitude faster than those requiring training.

Weight agnostic neural networks One of pioneering works in zero-shot NAS is presented by Gaier & Ha (2019). They demonstrate a constructor that builds up neural architectures based on the mean accuracy over several initialisations with constant shared weights and the number of parameters contained within the model. They report that the resulting model achieves $82.0\% \pm 18.7\%$ on MNIST data (LeCun et al., 2010) with random weights at initialisation, and over 90% when the weights are fixed to the best performing constants. While these results are very intriguing, the authors admit that these architectures do not perform particularly well upon training. Moreover, back in 2019 the benchmark databases of trained architectures that are routinely used now to compare NAS metrics with each other were not released yet, which disables us to compare this zero-shot method against the most recent ones.

Jacobian covariance In 2020, Mellor et al. (2020) present another metric, which employs the fact that rectified linear unit (ReLU, Agarap (2018)) activation functions lead to distinct activation patterns among architectures. Concretely, every image from a single mini-batch yields a binary activation vector. Together these vectors form a binary matrix, and the logarithm of its determinant serves as a scoring metric. Following the established notation from the NAS community, we refer to this metric as `jacov`[1]. Authors show that larger `jacov` values are associated with better training performances, which leads to a conclusion that high performing networks should be able to distinguish the inputs before being trained. This important takeaway comes with a considerable drawback: the method can only be implemented on networks with ReLU activation functions, which limits its applicability to convolutional architectures.

Coefficient of variance Another early work on fully trainless NAS belongs to Gracheva (2021), who evaluates the stability of untrained scores over random weights initialisations. The author initialises the networks with multiple random seeds, and architectures are selected based on the coefficient of variance `CV` of the accuracy at initialisation. While `CV` performance is associated with high error rate, the author makes a conclusion that a good architecture should be stable against random weight fluctuations. While this method can in theory be applied to any neural architecture type, it requires multiple initialisations and is relatively heavy comparing to `jacov` and later methods. Furthermore, accuracy-based scoring metrics can only be applied to classification problems, and it is not clear how to extend `CV` implementation to the regression tasks.

Pruning-at-initialisation proxies Several powerful zero-cost proxy have emerged as an adaptation of pruning-at-initialisation methods to NAS in the work by Abdelfattah et al. (2021): `grad_norm` (Wang et al., 2020), `snip` (Lee et al., 2018), `synflow` (Tanaka et al., 2020). These metrics are originally developed to evaluate the salience of network's parameters, with the aim to prune away potentially meaningless synapses prior to training in order to reduce its cost. They require a single forward-backward pass to compute the loss. Then, the importance of parameters is computed as multiplication of the weight value and gradient value. Abdelfattah et al. (2021) integrate the salience over all the parameters in the network to evaluate its potential upon training. What is particularly interesting about the `synflow` metric is that it evaluates the architectures without looking at the data, by computing the loss as the product of all the weights' values (which are randomly initialised). `synflow` metric show the most consistent performance among various search spaces, and sets the state-of-the-art for the zero-cost NAS.

[1]Anecdotally, in their first version of the paper released in June 2020, the authors presented another scoring method using Jacobian covariance, and achieved significantly different performance. The metric was updated several months later, but the nickname `jacob_cov` persisted and was recently shortened down to `jacov`.

Both `jacov` and `synflow` do not depend on true labels, which arguably reduces the effect of the human error during data labelling. Moreover, `jacov` does not require gradient computation, which renders this method less memory-intensive.

The above results imply that neural networks might have some intrinsic property which defines their prediction potential prior to training. Such property should not depend on the values of trainable parameters (weights), but only on network's topology. In the present work, we combine the takeaways from the existing trainless NAS implementations to present a new metric, which significantly outperforms existing zero-cost NAS methods.

1.2 NAS BENCHMARKS

To guarantee the reproducibility of our metric and to compare its performance against other NAS algorithms, we evaluate it on the three widely used NAS benchmark datasets.

NAS-Bench-101 The first and one of the largest NAS benchmarks sets of trained architectures. It consists of over $423k$ cell-based convolutional neural networks (Ying et al., 2019). The architectures are made of three stacks of cells each followed by max-pooling layers. Cells may have up to 7 vertices and 9 edges, with 3 possible operations. This benchmark is trained multiple times on a single dataset, CIFAR-10 (Krizhevsky et al., 2009), with a fixed set of hyperparameters for 108 epochs.

NAS-Bench-201 This is a set of architectures with fixed skeleton, consisting of convolution layer and three stacks of cells, connected by a residual block (Dong & Yang, 2020). Each cell is a densely-connected directed acyclic graph with 4 nodes, 5 possible operations and no limits on the number of edges, providing a total of $15,625$ possible architectures. Architectures are trained on three major datasets: CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009) and a downsampled version of ImageNet (Chrabaszcz et al., 2017). Training hyperparameters are fixed, and the training spans 200 epochs.

NAS-Bench-NLP As the name suggests, this benchmark is focused on architectures suitable for neural language processing (Klyuchnikov et al., 2022). Concretely, it consists of randomly-generated recurrent neural networks. Recurrent cells are comprised of 24 nodes, 3 hidden states and 3 input vectors *at most*, with 7 allowed operations. Here, we only consider models trained evaluated on Penn Treebank (PTB, Marcinkiewicz (1994)) dataset: $14,322$ random networks with a single seed and $4,114$ with tree seeds. Training was conducted with fixed set of hyperparameters and the perplexities are reported for the mark of 50 epochs.

2 EPSILON METRIC

2.1 INCEPTION

The metric that we share in the present work is inspired by two existing NAS methods: `CV` (Gracheva, 2021) and weight agnostic neural networks (Gaier & Ha, 2019). Both metrics aim to exclude the values of individual weight from consideration when valuating networks: the former cancels out the individual weights via multiple random initialisations, while the latter sets them to the same value across the network. It is indeed very intriguing to see that architectures can be characterised purely by their topology.

As it was mentioned above, `CV` metric has two principal disadvantages. While it shows quite consistent trend with trained accuracy, it clearly suffers from high uncertainty. This can be partly explained by the fact that random weights initialisations bring in some noise. Our idea is that replacing random initialisations with constant shared weight initialisations, similarly to Gaier & Ha (2019), should improve the method's performance.

The second weak point is that the metric is developed for classification problems and is based on accuracy, and it is not clear how to apply this metric to regression problems. Coefficient of variation is a ratio of standard deviation to mean, and Gracheva (2021) shows that `CV` metric shows negative correlation with train accuracy. This implies that the mean untrained accuracy should be maximised. On the other hand, for regression tasks the performance is typically computed as some kind of error, which is sought to be minimised. It is not obvious whether division by mean untrained *error* result in the same trend for `CV` metric.

Algorithm 1 Algorithm for `epsilon` zero-cost metric computation

```
Select a batch of data from train set
for arch in search search do
    Initialise empty output matrix
    for weight in [val1, val2] do
        Initialise arch with constant shared weight
        Forward pass the batch through arch
        Get and flatten outputs
        Minmax normalise outputs (Eq. 1)
        Append outputs to the output matrix
    end for
    Compute difference between the rows of the output matrix (Eq. 2)
    Compute mean over the output matrix (Eq. 3)
    Compute epsilon metric (Equation 4)
end for
```

To address this issue we decided to consider raw outputs. This modification renders the method applicable to any kind of neural architecture. Yet it comes with a difference: accuracy returns a single value per batch of data, while raw outputs are $[N_{\text{BS}} \times L]$ matrices, where N_{BS} is the batch size and L is the length of a single output.[2] We flatten these matrices to obtain a single vector $\boldsymbol{v}$ of length $L_v = N_{\text{BS}} \times L$ per initialisation. We then stuck both initialisations into a single output matrix $\boldsymbol{V}$.

Before proceeding to statistics computation over initialisations, we also must normalise the output vectors: in the case of fixed weights, outputs scale with weight values. In order to compare initialisations on a par with each other, we use min-max normalisation:

$$\boldsymbol{V}_i' = \frac{\boldsymbol{V}_i - \min(\boldsymbol{V}_i)}{\max(\boldsymbol{V}_i) - \min(\boldsymbol{V}_i)}, \tag{1}$$

where i is the index for initialisations, $i \in [0, 1]$.

We noticed that two distinct weights are sufficient to grasp the difference between initialisations. Accordingly, instead of standard deviation we use mean absolute error between the normalised outputs of two initialisations:

$$\delta = \frac{1}{L_v} \sum_{j=0}^{L_v} |\boldsymbol{V}_{1,j}' - \boldsymbol{V}_{2,j}'|. \tag{2}$$

The mean is computed over the outputs of both initialisations as following:

$$\mu = \frac{1}{L_v} \sum_{j=0}^{L_v} \frac{\boldsymbol{V}_{1,j}' + \boldsymbol{V}_{2,j}'}{2} = \frac{1}{2L_v} \sum_{i=0}^{2} \sum_{j=0}^{L_v} \boldsymbol{V}_{i,j}' \tag{3}$$

Finally, the metric is computed as the ratio of δ and μ:

$$\varepsilon = \frac{\delta}{\mu}. \tag{4}$$

We refer to our metric as `epsilon`, as a tribute to ε symbol used in mathematics to denote error bounds. The algorithm for the `epsilon` computation is given in Algorithm 1.

[2]This length depends on the task and/or architecture: for regression tasks $L = 1$, for classification L is equal to the number of classes, and for recurrent networks it depends on the desired length of generated string.

3 RESULTS AND DISCUSSION

3.1 EMPIRICAL VALUATION

Here we evaluate the performance of `epsilon` and compare it to the results for other major zero-cost NAS metrics reported in Abdelfattah et al. (2021). We use the following evaluation scores (computed with `NaN` omitted):

- Spearman ρ (global): Spearman rank correlation ρ evaluated on the entire dataset.
- Spearman ρ (top-10%): Spearman rank correlation ρ for the top-10% performing architectures.
- Kendall τ (global): Kendall rank correlation coefficient τ evaluated on the entire dataset.
- Kendall τ (top-10%): Kendall rank correlation coefficient τ for the top-10% performing architectures.
- Top-10%/top-10%: fraction of top-10% performing models within the top-10% models ranked by zero-cost scoring metric (%).
- Top-64/top-10%: number of top-64 models ranked by zero-cost scoring metric within top-5% performing models.

Table 1: Zero-cost metrics performance evaluated on NAS-Bench-201 search space and its three datasets: CIFAR-10, CIFAR-100 and ImageNet16-120. Values from Abdelfattah et al. (2021) are given for reference between brackets.

Metric	Spearman ρ		Kendall τ		Top-10%/	Top-64/	
	global	top-10%	global	top-10%	top-10%	top-5%	
CIFAR-10							
`synflow`	0.74	0.18	0.54	0.12	45.75 46	29	(44)
`grad_norm`	0.59 (0.58)	−0.36 (−0.38)	0.43	−0.21	30.26 30	1	(0)
`grasp`	0.51 (0.48)	−0.35 (−0.37)	0.36	−0.21	30.77 30	3	(0)
`snip`	0.60 (0.58)	−0.36 (−0.38)	0.44	−0.21	30.65 31	1	(0)
`fisher`	0.36	−0.38	0.26	−0.24	4.99 5	0	(0)
`jacov`	−0.73 (0.73)	0.15 (0.17)	0.55	−0.10	24.72 25	11	(15)
`epsilon`	0.87	0.55	0.70	0.40	67.39	59	
CIFAR-100							
`synflow`	0.76	0.42	0.57	0.29	49.71 50	45	(54)
`grad_norm`	0.64	−0.09	0.47	−0.05	35.00 35	0	(4)
`grasp`	0.55 (0.54)	−0.10 (−0.11)	0.39	−0.06	35.32 34	3	(4)
`snip`	0.64 (0.63)	−0.08 (−0.09)	0.47	−0.05	35.25 36	0	(4)
`fisher`	0.39	−0.15 (−0.16)	0.28	−0.10	4.22 4	0	(0)
`jacov`	−0.70 (0.71)	0.07 (0.08)	0.54	0.05	22.11 24	7	(15)
`epsilon`	0.90	0.59	0.72	0.43	81.24	62	
ImageNet16-120							
`synflow`	0.75	0.55	0.56	0.39	43.57 44	26	(56)
`grad_norm`	0.58	0.12 (0.13)	0.43	0.09	31.29 31	0	(13)
`grasp`	0.55 (0.56)	0.10	0.39	0.07	31.61 32	2	(14)
`snip`	0.58	0.13	0.43	0.09	31.16 31	0	(13)
`fisher`	0.33	0.02	0.25	0.01	4.61 5	0	(0)
`jacov`	0.70 (0.71)	0.08 (0.05)	0.53	0.05	29.63 44	10	(15)
`epsilon`	0.85	0.53	0.67	0.37	71.51	59	

Comparing the results for `epsilon` with other zero-cost NAS metrics, we can see that it outperforms them by a good margin. The illustrations (Figure 1, 5) further confirm the applicability of the method to the NAS-Bench-201 field. However, NAS-Bench-201 is a relatively compact search space, furthermore, it has been used for `epsilon` development.

3.1.1 NAS-BENCH-201

The results for `epsilon` performance on NAS-Bench-201 are given in Table 1 along with other zero-cost NAS metrics. The Kendall τ score is not reported in Abdelfattah et al. (2021), but it is considered to be more robust than Spearman ρ and being increasingly used for NAS metric evaluation. We use the data provided by Abdelfattah et al. (2021) to evaluate their Kendal τ. For some evaluation scores, the results that we obtain differ from the original paper. In such cases, we report our results and indicate the original values between brackets. In particular, there is a discrepancy in computing the values in the last column, *Top-64/top*-10%, while the rest of the results are consistent. We have double-checked our calculations and Figure 5 (Appendix) suggest that our calculations are correct.

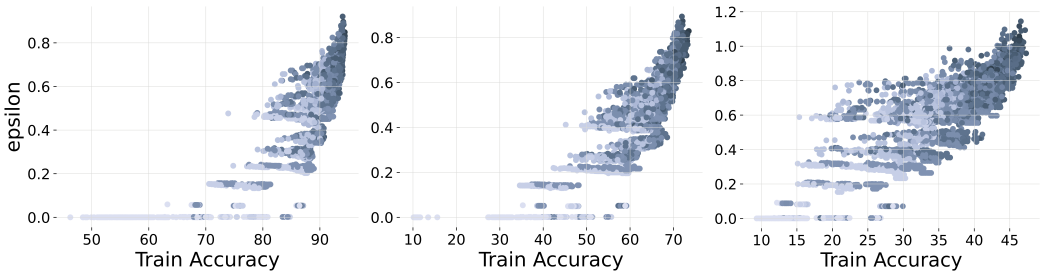

Figure 1: Zero-cost NAS `epsilon` metric performance illustration for NAS-Bench-201 search space evaluated on CIFAR-10, CIFAR-100 and ImageNet16-120 datasets. Each point represents an architecture with colours representing the number of parameters – the darker the more parameters in the network. Figure represents search space of $15,625$ networks (excluding architectures with `NaN` scores).

3.1.2 NAS-BENCH-101

We use NAS-Bench-101 space to confirm that the success of `epsilon` metric in the previous section is not due to overfitting to the NAS-Bench-201 search space, and to see how its applies to a vaster search space. Table 2 together with Figure 2 confirm that it works well on NAS-Bench-201, too.

3.2 NAS-BENCH-NLP

Both NAS-Bench-201 and NAS-Bench-101 are created to facilitate NAS in the field of image recognition. They operate convolutional networks of very similar constitution. To truly probe generalisability of the `epsilon` metric we test it on NAS-Bench-NLP. Both input data format and architecture type are different from he first two search spaces.

Unfortunately, there are no data available for NAS-Bench-NLP in Abdelfattah et al. (2021). Consequently, we could not reproduce their results. Therefore, in Table 3 we give only values provided in the paper together with our `epsilon` metric. We would like to note that unlike accuracy, perplexity used for language-related ML problems should be minimised. Therefore, the signs of correlations with scoring metrics should be reversed, which is not the case for numbers given in Abdelfattah et al. (2021).

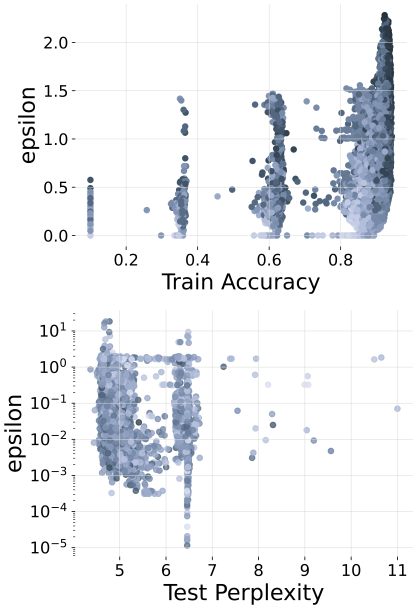

Figure 2: Zero-cost NAS `epsilon` metric performance on NAS-Bench-101 search space, CIFAR-10 dataset (top) and NAS-Bench-NLP search space, PTB dataset (bottom) . Each point corresponds to an architecture, the darker the colour the more parameters it contains. Figure shows $423,624$ and $14,322$ networks for NAS-Bench-101 and NAS-Bench-NLP, respectively (excluding architectures with `NaN` scores).

The performance of epsilon metric on the NAS-Bench-NLP space is not exceptional. While there is a trend towards better architectures with increasing metric value, the level of noise is beyond acceptable. This might stem from the characteristics of the benchmark itself (factors like relatively small sample of networks given large space, chosen hyperparameters, dropout rates, *etc* may distort NAS performance). Nonetheless, the trend is visible enough to conclude that epsilon metric can be applied to recurrent type architectures.

Table 2: Zero-cost metrics performance evaluated on NAS-Bench-101 search space, CIFAR-10 dataset. Values from Abdelfattah et al. (2021) are given for reference between brackets.

Metric	Spearman ρ		Kendall τ		Top-10%/	Top-64/	
	global	top-10%	global	top-10%	top-10%	top-5%	
CIFAR-10							
synflow	0.37	0.14	0.25	0.10	22.67 23	4	(12)
grad_norm	−0.20	−0.05 (0.05)	−0.14	−0.03	1.98 2	0	(0)
grasp	0.45	−0.01	0.31	−0.01	25.60 26	0	(6)
snip	−0.16	0.01 (−0.01)	−0.11	0.00	3.43 3	0	(0)
fisher	−0.26	−0.07 (0.07)	−0.18	−0.05	2.65 3	0	(0)
jacov	0.38 (0.38)	−0.08 (0.08)	−0.05	0.05	1.66 2	0	(0)
epsilon	0.62	0.12	0.44	0.08	40.33	10	

Table 3: Zero-cost metrics performance evaluated on NAS-Bench-NLP search space, PTB dataset.

Metric	Spearman ρ		Kendall τ		Top-10%/	Top-64/
	global	top-10%	global	top-10%	top-10%	top-5%
PTB						
synflow	0.34	0.10	—	—	22	—
grad_norm	−0.21	0.03	—	—	10	—
grasp	0.16	0.55	—	—	4	—
snip	−0.19	−0.02	—	—	10	—
fisher	|	|	—	—	—	—
jacov	0.38	0.04	—	—	38	—
epsilon	0.34	−0.16	0.23	−0.14	3.57	3

3.3 Integration with other NAS methods

While it is possible to utilise zero-cost metrics on their own, usually they are implemented within other NAS. Here we provide simple examples of random search and ageing evolution algorithms when used together with epsilon.

Similarly to Abdelfattah et al. (2021), we compare random search performance with and without warm-up. During the warm-up session, initial pool of $3,000$ architectures is formed, and the first 64 steps algorithm updates the best test performance based from this pool, starting with the networks with highest epsilon score.

For ageing evolution, the same principle is applied. En plus, we report the results of implementation where every next parent is decided based on the highest epsilon score (move), with and without warm-up Then, the child is created by parent mutation (within edit distance of 1) and added to pool.

For both NAS algorithms, we run procedure until the number of trained architectures reaches 300, and perform 100 random rounds. Figure 3 shows that epsilon metrics leads to considerable improvements, both in terms of time and precision. Unsurprisingly, the best setup during ageing evolution is warm-up with move.

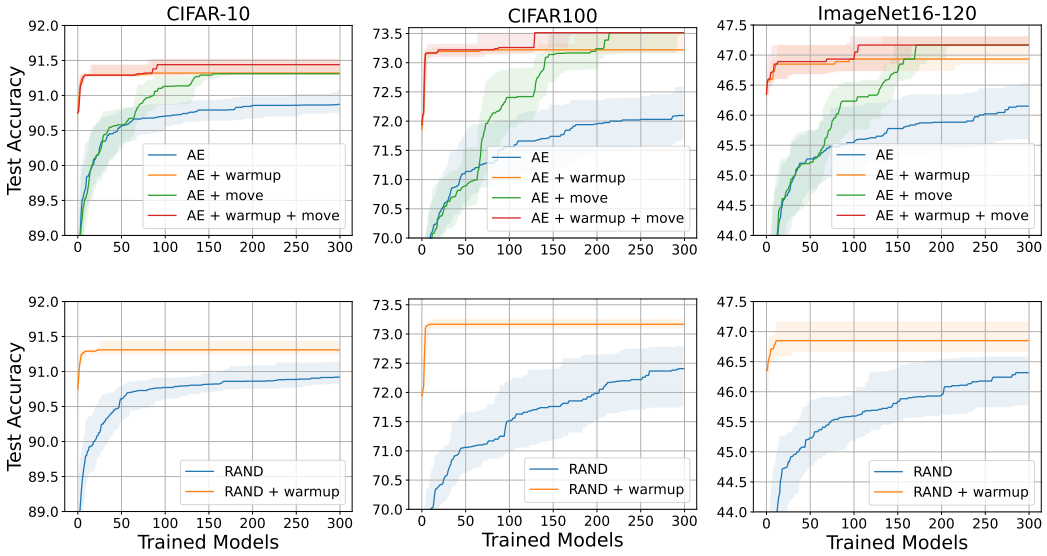

Figure 3: `epsilon` integration within ageing evolution (top) and random search (bottom) NAS algorithms for three datasets from NAS-Bench-201 search space.

4 Ablation Studies

While our metric is quite straightforward to implement, it does require several hyperparameters to be set. In this section, we provide the results of the ablations studies and analyse their results.

4.1 Weights

To compute the epsilon metric we initialise the networks with two distinct constant shared weights. The question is: how do we choose their values, and how much does this choice affect the efficiency of the whole method?

To answer this question, we ran a series of tests with multiple pairs of weights. In pairs, it makes sense to set the first weight always greater than the second one. This consideration stems from the fact that epsilon is based on mean absolute difference between two initialisations. Therefore, it will output the same score for $[w_1, w_2]$ and $[w_2, w_1]$ combinations, and will be exactly zero in case where the two weights are equal. Tables 8, 8, 9 summarises the results of our tests (see Appendix).

The results show that there is indeed some variation between different initialisations. There are two effects that impact the performance. Firstly, the greater is the difference between the two weights, the higher is the correlation between `epsilon` and trained performance. This makes sense, since if the weights are too close to each other the difference between the outputs becomes minimal, and distinction between architectures in terms of `epsilon` is no longer possible.

Secondly, we can see that too low and too high weights results in lower number of architectures with non-`NaN` scores. Extreme weights result in all zeros or infinite output values and, consequently, in `NaN` value of the epsilon metric. Naturally, the deeper the network is, the more there is of signal attenuation, and the higher is the probability of `NaN` score. We remind that prior to `epsilon` computation, the outputs are normalised to the same range $[0, 1]$, which means that the metric itself does not become smaller for deeper networks. This extinction effect is due to the limited sensitivity of `float` type. Theoretically, with infinite float sensitivity, there would be no extinction and hence no effects of the choice of the weights.

As the rule of thumb, for every new ML problem we suggest to run `epsilon` evaluation on a subset of architectures with several weights, and select the minimum and maximum weights that do not cause excessive `NaN` outputs. While we acknowledge that this procedure is quite subjective, importantly it does not require any training.

Weight combinations that we selected for each search space are given in Table 4.

4.2 BATCH SIZE

To test the sensitivity of the method against the batch size, we ran the scoring routine with 8 different batch sizes $[8, 16, 32, 64, 128, 512, 1024]$, each with 10 random batches. Figures show that as expected the larger the batch size, the better and more stable is performance of `epsilon` metric. There is almost no improvement for batch sized over 256, and in our experiments we set it to this value.

Table 4: Optimal weights for `epsilon` evaluation for three search spaces and their datasets.

Search space	Dataset	Optimal weights
NAS-Bench-101	CIFAR-10	$[10^{-4}, 10]$
NAS-Bench-201	CIFAR-10	$[10^{-7}, 1]$
	CIFAR-100	$[10^{-7}, 1]$
	ImageNet16-120	$[10^{-7}, 1]$
NAS-Bench-NLP	PTB	$[10^{-5}, 10^{-5}]$

4.3 EMBEDDING INITIALISATION

Care should be taken when initialising the networks containing embedding layer: if embedding is initialised with all constants, there is no difference between the embedded input. The performance of our metric in this case will be analogous to that with batch size of 1.

Our ablation studies, reported in detail in Appendix A.2, show that as soon as the embedding is initialised with any non-constant weights, it practically does not influence the final results.

5 CONCLUSIONS

In this work, we present a novel zero-cost NAS scoring metric `epsilon`. It is reflects how the different are the outputs of neural networks depending on the value of the constant shared weight used at initialisation. It shows that the higher is this difference, the better will the network perform upon training. In other words, `epsilon` probes networks sensitivity to varying weights magnitudes.

Our method requires only two initialisations followed by two forward passes. It does not require true labels, gradient computation, and according to our tests can even be used with synthetic data (see Section A.4. The evaluation of the metric takes $0.1 \sim 1$ seconds per architectures on a GPU (depending on the size of architecture and batch size), but can be easily realised on a CPU. `epsilon` is not limited to any specific neural network type, in other words, it can be implemented for any ML problem (care should be taken for embedding initialisation, as explained in Section 4.3).

Our metric is evaluated on three staple NAS search spaces: NAS-Bench-201, NAS-Bench-101 and NAS-Bench-NLP. It shows stable good performance with each of them, regardless the data set. It also brings significant improvements when implemented with random and evolutionary NAS algorithms (see Section 3.3).

The only factor that affects `epsilon` metric performance is the choice of the values for the constant weights during initialisation. Our tests show that it must be setup individually for each search space. For each search space, the weights are set so that they span the maximum range of orders of magnitude, under condition that they do not lead to excessive signal extinction (different architectures show different sensitivities to big and small weights). We are not completely satisfied with this heuristic and hope to automate the process of weights selection in our future work.

REPRODUCIBILITY STATEMENT

The codes that reproduce all the results in this paper can be found at the following HTML: https://anonymous.4open.science/r/EpsilonNAS-FB1B/. Our results are produced with DGX station (4 NVIDIA V100 GPUs).

ACKNOWLEDGMENTS

The authors would like to thank Drs. N and L for their continuous support and advice.

REFERENCES

Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D Lane. Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*, 2021.

Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: robustly stepping out of performance collapse without indicators. *arXiv preprint arXiv:2009.01027*, 2020.

Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3681–3690, 2019a.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1761–1770, 2019b.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

Adam Gaier and David Ha. Weight agnostic neural networks. *Advances in neural information processing systems*, 32, 2019.

Ekaterina Gracheva. Trainless model performance estimation based on random weights initialisations for neural architecture search. *Array*, 12:100082, 2021. ISSN 2590-0056. doi: https://doi.org/10.1016/j.array.2021.100082.

Roxana Istrate, Florian Scheidegger, Giovanni Mariani, Dimitrios Nikolopoulos, Constantine Bekas, and Adelmo Cristiano Innocenza Malossi. Tapas: Train-less accuracy predictor for architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3927–3934, 2019.

Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, pp. 367–377. PMLR, 2020.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Using Large Corpora*, 273, 1994.

Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. *arXiv preprint arXiv:2006.04647v1*, 2020.

Curtis G Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. *arXiv preprint arXiv:2103.14749*, 2021.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10293–10301, 2021.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114. PMLR, 2019.

A Appendix

A.1 Batch size ablation study

Here are the results of the batch size ablations. The tests are done on NAS-Bench-201 search space with its three datasets. For each batch we report medians, together with 25 and 75 percentiles, over 10 runs. We evaluate `epsilon` on $5,000$ architectures.

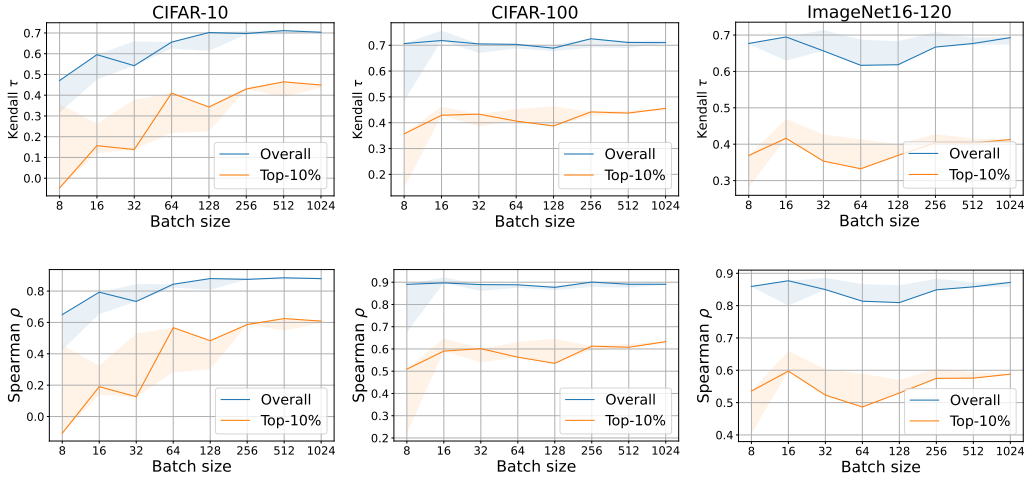

Figure 4: Batch size ablation study.

A.2 EMBEDDING ABLATION STUDY

In order to verify the effect of the embedding initialisation, we have run tests with 6 different initialisations:

- uniform positive 0.1: random uniform from ranges [0, 0.1]
- uniform positive 1: random uniform from ranges [0, 1]
- uniform centred 0.1: random uniform from ranges [-0.1, 0.1]
- uniform centred 1: random uniform from ranges [-1, 1]
- random 0.1: random normal centred at 0 with standard deviation of 0.1
- random 1: random normal centred at 0 with standard deviation of 1

Table 5 summarise our results of the embedding ablations. Ablation is done on NAS-Bench-NLP search space, PTB dataset (the only search space implementing embedding).

Table 5: Embedding ablation studies. Metric computed over 5000 initial architectures. Number of remaining architectures is given in the second column.

Embedding	Archs	Spearman ρ		Kendall τ		Top-10%/	Top-64/
		global	top-10%	global	top-10%	top-10%	top-5%
uniform positive 0.1	782	−0.38	0.17	−0.26	0.15	3.80	2
uniform positive 1	776	−0.37	0.21	−0.26	0.18	2.56	2
uniform centered 0.1	783	−0.40	0.18	−0.28	0.15	2.53	2
uniform centered 1	782	−0.46	0.18	−0.33	0.16	1.27	1
random 0.1	783	−0.42	0.18	−0.29	0.15	2.53	2
random 1	782	−0.47	0.19	−0.33	0.16	1.27	1

A.3 VISUALISATION OF OTHER ZERO-COST NAS METRICS

In the work of Abdelfattah et al. (2021) the metrics are presented through statistical measures, but we feel that visualisation helps to improve understanding. Here we provide visualisations for two search spaces built on the data provided by authors.

A.4 SYNTHETIC DATA

In this section we test the importance of the input data by feeding our metric several synthetic input data on CIFAR-10. We feed networks with 5 types of data:

- real data: batch of CIFAR-10 data
- grey scale images: images within the batch are solid colour ranging from black to white
- random normal: images are filled with random values following normal distribution with $[\mu, \sigma] = [0, 1]$
- random uniform: images are filled with random values following uniform distribution with $[\mu, \sigma] = [-1, 1]$
- random uniform (+): images are filled with random values following uniform distribution with $[min, \max] = [0, 1]$

All the tests are performed with batch size of 256, weights $[10^{-7}, 1]$ and 5000 architectures. Table 6 shows that even though the performance with synthetic data drops comparing to real data, it is still very good.

Curiously, grey scale images, which are filled with constant values, show the closest results to the real data. Also, `epsilon` metric with grey scale data outperforms `synflow`. This is an important achievement, because `synflow` does not use input data, and is therefore data independent. Our results show that `epsilon` has a potential to be used with no data whatsoever.

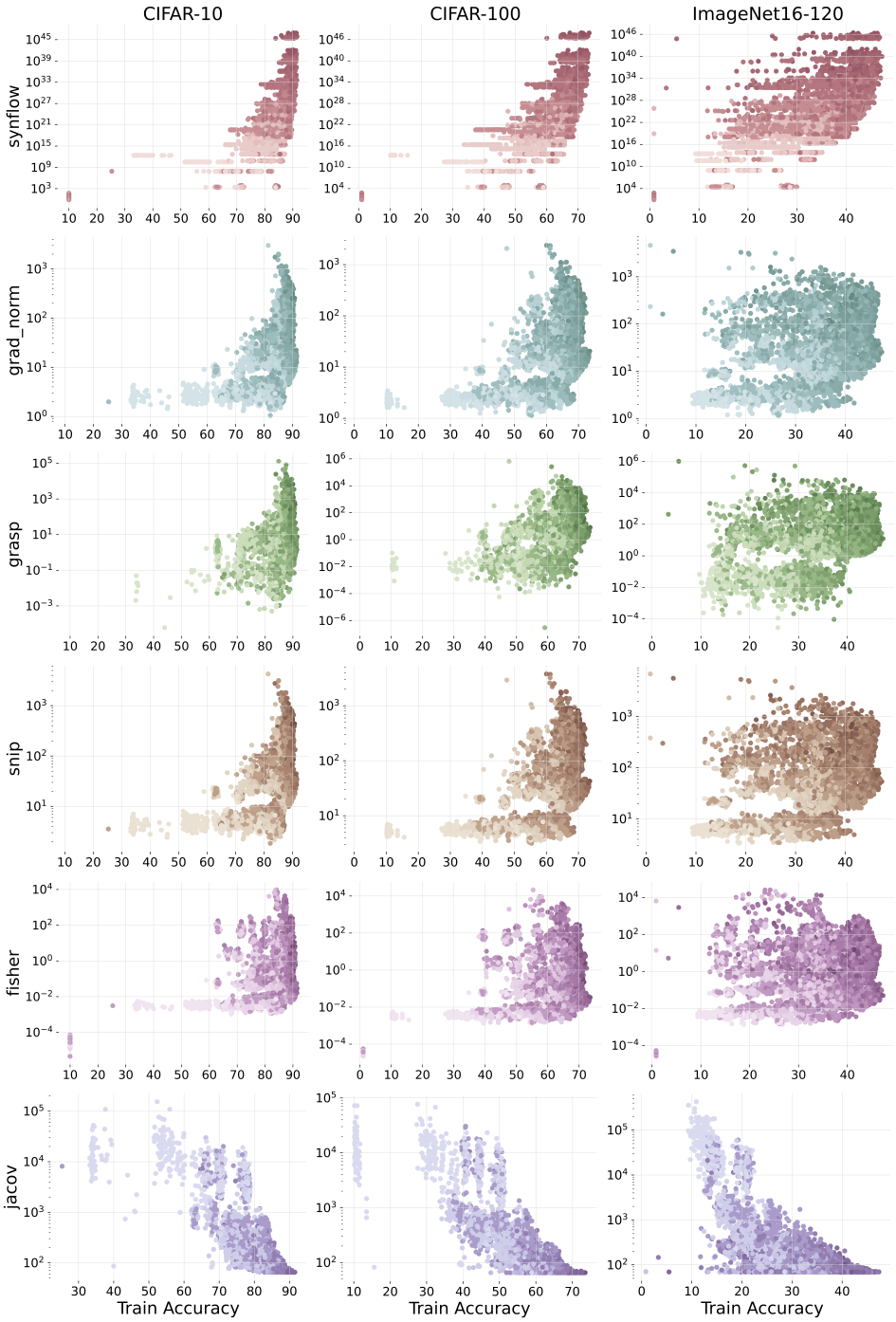

Figure 5: Zero-cost metrics performance illustration for NAS-Bench-201 search space evaluated on CIFAR-10, CIFAR-100 and ImageNet16-120 datasets, based on data from Abdelfattah et al. (2021). Each point represents an architecture with colours representing the number of parameters – the darker the more parameters in the network. Figure represents search space of $15,625$ networks (excluding architectures with `NaN` scores).

Table 6: Synthetic data tests with CIFAR-10 dataset, NAS-Bench-201 search space. Tests are based on evaluation of 5000 architectures.

Metric	Spearman ρ		Kendall τ		Top-10%/	Top-64/
	global	top-10%	global	top-10%	top-10%	top-5%
real data	0.87	0.59	0.70	0.43	65.88	45
grey scale	0.87	0.44	0.68	0.31	59.46	34
random normal	0.54	0.15	0.38	0.14	14.97	3
random uniform	0.56	0.17	0.40	0.16	14.19	3
random uniform (+)	0.61	0.17	0.43	0.16	16.22	4

A.5 Optimal weights search

In this section, we print the tables for weights ablation study. There are three of them – one per search space. For NAS-Bench-201 search space, tables are very similar between the datasets, and the minor differences do not affect the conclusions drawn from this data.

Table 7: Zero-cost metrics performance evaluated on NAS-Bench-101 search space, CIFAR-10 dataset. Number of architectures with non-`NaN` values are given in the second column.

Weights	Archs	Spearman ρ		Kendall τ		Top-10%/	Top-64/
		global	top-10%	global	top-10%	top-10%	top-5%
0.0001, 0.001	2120	0.47	0.41	0.33	0.28	38.21	23
0.0001, 0.01	2120	0.61	0.03	0.43	0.02	41.98	24
0.0001, 0.1	2120	0.61	0.03	0.43	0.02	41.98	24
0.0001, 1	2120	0.61	0.03	0.43	0.02	41.98	24
0.0001, 10	2120	0.61	0.03	0.43	0.02	41.98	24
0.001, 0.01	4968	0.29	−0.05	0.20	−0.03	17.30	10
0.001, 0.1	4968	0.29	−0.05	0.19	−0.03	17.30	8
0.001, 1	4968	0.29	−0.05	0.19	−0.03	17.30	8
0.001, 10	4968	0.29	−0.05	0.19	−0.03	17.30	8
0.01, 0.1	5000	0.21	−0.12	0.14	−0.08	4.60	0
0.01, 1	5000	0.21	−0.12	0.14	−0.08	4.60	0
0.01, 10	5000	0.21	−0.12	0.14	−0.08	4.60	0
0.1, 1	5000	−0.47	0.04	−0.32	0.03	0.20	0
0.1, 10	5000	−0.45	0.00	−0.30	0.00	0.20	0
1, 10	5000	−0.44	0.01	−0.29	0.00	0.20	0

Table 8: Zero-cost metrics performance evaluated on NAS-Bench-201 search space, CIFAR-10 dataset. Number of architectures with non-`NaN` values are given in the second column.

Weights	Archs	Spearman ρ		Kendall τ		Top-10%/	Top-64/
		global	top-10%	global	top-10%	top-10%	top-5%
1e-08, 1e-07	2957	0.27	−0.46	0.20	−0.37	0.00	3
1e-08, 1e-06	2957	0.42	−0.41	0.28	−0.30	4.39	1
1e-08, 1e-05	2957	0.43	−0.19	0.31	−0.13	9.12	2
1e-08, 0.0001	2957	0.47	−0.01	0.32	0.00	9.12	2
1e-08, 0.001	2957	0.62	0.06	0.43	0.05	18.92	3
1e-08, 0.01	2957	0.83	0.69	0.64	0.49	68.40	56
1e-08, 0.1	2957	0.84	0.64	0.65	0.46	68.03	56
1e-08, 1	2957	0.87	0.59	0.70	0.43	65.88	45
1e-08, 10	2957	0.87	0.58	0.70	0.43	66.78	46
1e-07, 1e-06	2957	0.33	−0.20	0.22	−0.13	4.76	1
1e-07, 1e-05	2957	0.39	−0.06	0.29	−0.04	9.15	2
1e-07, 0.0001	2957	0.47	−0.01	0.32	0.00	9.12	2
1e-07, 0.001	2957	0.62	0.06	0.43	0.05	18.92	3
1e-07, 0.01	2957	0.83	0.69	0.64	0.49	68.40	56
1e-07, 0.1	2957	0.84	0.64	0.65	0.46	68.03	56
1e-07, 1	2957	0.87	0.59	0.70	0.43	65.88	45
1e-07, 10	2957	0.87	0.58	0.70	0.43	66.78	46
1e-06, 1e-05	2957	0.41	−0.06	0.30	−0.04	9.15	2
1e-06, 0.0001	2957	0.47	−0.01	0.32	0.00	9.12	2
1e-06, 0.001	2957	0.62	0.06	0.43	0.05	18.92	3
1e-06, 0.01	2957	0.83	0.69	0.64	0.49	68.40	56
1e-06, 0.1	2957	0.84	0.64	0.65	0.46	68.03	56
1e-06, 1	2957	0.87	0.59	0.70	0.43	65.88	45
1e-06, 10	2957	0.87	0.58	0.70	0.43	66.78	46
1e-05, 0.0001	2957	0.47	−0.01	0.32	0.00	9.12	2
1e-05, 0.001	2957	0.62	0.06	0.43	0.05	18.92	3
1e-05, 0.01	2957	0.83	0.69	0.64	0.49	68.40	56
1e-05, 0.1	2957	0.84	0.64	0.65	0.46	68.03	56
1e-05, 1	2957	0.87	0.59	0.70	0.43	65.88	45
1e-05, 10	2957	0.87	0.58	0.70	0.43	66.78	46
0.0001, 0.001	2957	0.62	0.06	0.43	0.05	19.26	3
0.0001, 0.01	2957	0.82	0.69	0.63	0.49	68.42	56
0.0001, 0.1	2957	0.84	0.64	0.65	0.46	68.14	56
0.0001, 1	2957	0.87	0.58	0.69	0.42	65.88	46
0.0001, 10	2957	0.87	0.58	0.69	0.42	67.24	61
0.001, 0.01	4802	0.25	0.41	0.17	0.28	30.35	52
0.001, 0.1	4802	0.31	0.27	0.20	0.19	26.82	25
0.001, 1	4802	0.31	0.22	0.21	0.15	26.26	6
0.001, 10	4802	0.26	0.21	0.17	0.15	26.82	7
0.01, 0.1	4907	0.15	0.42	0.10	0.29	37.47	57
0.01, 1	4907	0.12	0.39	0.06	0.27	35.85	57
0.01, 10	4907	0.12	0.39	0.06	0.27	35.85	56
0.1, 1	4907	0.01	0.26	−0.03	0.16	0.84	0
0.1, 10	4907	0.01	0.26	−0.03	0.16	0.43	0
1, 10	4907	−0.02	0.25	−0.05	0.16	0.00	0

Table 9: Zero-cost metrics performance evaluated on NAS-Bench-NLP search space, PTB dataset. Number of architectures with non-`NaN` values are given in the second column.

Weights	Archs	Spearman ρ		Kendall τ		Top-10%/	Top-64/
		global	top-10%	global	top-10%	top-10%	top-5%
1e-07, 1e-06	2393	-0.24	0.32	-0.16	0.22	0.00	0
1e-07, 1e-05	2391	-0.22	0.52	-0.15	0.39	1.67	0
1e-07, 0.0001	2388	-0.35	0.49	-0.24	0.37	0.42	1
1e-07, 0.001	2332	-0.42	0.50	-0.29	0.37	2.99	1
1e-07, 0.01	1215	-0.30	0.13	-0.21	0.09	4.20	1
1e-07, 0.1	585	-0.49	0.55	-0.36	0.44	1.69	1
1e-07, 1	403	-0.12	0.08	-0.08	-0.00	12.20	7
1e-06, 1e-05	3286	-0.11	0.49	-0.07	0.36	0.91	0
1e-06, 0.0001	3283	-0.24	0.47	-0.16	0.34	0.62	0
1e-06, 0.001	3226	-0.33	0.44	-0.23	0.33	4.64	1
1e-06, 0.01	1639	-0.26	0.13	-0.18	0.09	3.40	2
1e-06, 0.1	841	-0.41	0.13	-0.28	0.12	10.71	1
1e-06, 1	594	-0.09	0.13	-0.06	0.08	11.67	7
1e-05, 0.0001	3948	-0.32	0.30	-0.21	0.22	0.51	0
1e-05, 0.001	3892	-0.35	0.20	-0.24	0.16	3.85	2
1e-05, 0.01	1900	-0.32	0.27	-0.22	0.18	2.86	1
1e-05, 0.1	913	-0.38	0.31	-0.28	0.25	13.04	1
1e-05, 1	649	-0.18	0.36	-0.12	0.25	10.77	7
0.0001, 0.001	3987	-0.35	0.18	-0.24	0.14	3.76	0
0.0001, 0.01	1943	-0.37	0.38	-0.26	0.25	3.59	1
0.0001, 0.1	925	-0.43	0.26	-0.30	0.21	11.83	1
0.0001, 1	663	-0.26	0.35	-0.18	0.24	10.45	7
0.001, 0.01	1936	-0.32	0.40	-0.21	0.26	2.59	3
0.001, 0.1	918	-0.38	0.27	-0.30	0.22	8.70	2
0.001, 1	663	-0.23	-0.20	-0.19	-0.17	7.46	5
0.01, 0.1	910	-0.36	0.04	-0.25	0.02	8.79	2
0.01, 1	654	-0.25	0.06	-0.16	0.04	9.09	5
0.1, 1	652	-0.27	0.20	-0.19	0.13	10.61	7