Faster Performance Estimation for NAS with Embedding Proximity Score

Gideon Franken G.G.H.FRANKEN@STUDENT.TUE.NL
Prabhant Singh P.SINGH@TUE.NL
Joaquin Vanschoren J.VANSCHOREN@TUE.NL
Eindhoven University of Technology,
Groene Loper 5,
Eindhoven,
5600MB,
The Netherlands

Editor: Editor's name

Abstract

Neural Architecture Search methods generate large amounts of candidate architectures that need training to assess their performance and find an optimal architecture. To minimize the search time we use different performance estimation strategies. The effectiveness of such strategies varies in terms of accuracy and fit and query time. We propose Embedding proximity score (EmProx). EmProx builds a meta-model that maps candidate architectures to a continuous embedding space using an encoder-decoder framework. The performance of candidates is then estimated using weighted kNN based on the embedding vectors of architectures of which the performance is known. Performance estimations of this method are comparable to similar predictors in terms of accuracy while being nearly nine times faster to train compared to similar methods. Benchmarking against other performance estimation strategies currently used shows similar or better accuracy, while being five up to eighty times faster.

Code is made publicly available on GitHub[1].

1. Introduction

Neural architecture search (NAS) is the field of automatically designing novel neural network architectures for a specific task. The NAS problem is described as follows in White et al. (2021): given architectures a from search space $\mathcal{A}$, find the optimal architecture a^* as $a^* = argmax_{a \in \mathcal{A}} f(a)$, where f denotes the validation error of architecture a on a fixed dataset for a fixed number of epochs E. However, evaluating $f(a)$ requires significant computational resources. Hence, performance predictor f' is leveraged, defined as any function that predicts $f(a)$ without fully training a. Evaluating f' should take significantly less time than evaluating f and desirably $\{f'(a) \mid a \in \mathcal{A}\}$ has a high correlation with $\{f(a) \mid a \in \mathcal{A}\}$

We propose a new performance predictor f' called EmProx Score (Embedding Proximity Score) as a faster alternative for Neural Architecture Optimization (NAO) (Luo et al., 2018). Similar to NAO, EmProx uses an encoder-decoder framework to map candidate

1. https://github.com/openml-labs/EmProx

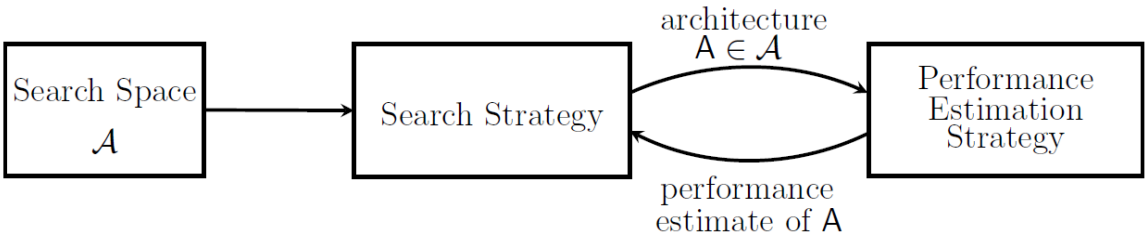

Figure 1: The three components of a Neural Architecture Search algorithm (Elsken et al., 2018). The search space defines the model architectures that can be discovered. The search strategy selects architectures from the search space to evaluate. To evaluate an architecture, a performance estimation strategy is used. Usually, this strategy returns the expected architecture's accuracy to the search strategy to further guide the architecture search.

architectures to a continuous embedding space. However, instead of training a multi-layer perceptron (MLP) on the embedding vectors to predict the accuracy of a candidate, EmProx uses *weighted k-nearest neighbors* on the embedding vectors. As such, EmProx estimates the performance of the query architecture based on the distance and accuracy of its nearest neighbors in the embedding space, which can be computed efficiently.

In the remainder of this paper, we first describe background and related work in Section 2. Section 3 describes the EmProx procedure in more detail, and Section 4 discusses implementation and evaluation details. Section 5 evaluates this method against a wide range of alternative approaches, and Section 6 concludes.

2. Related work

NAS methods consist of three components: the search space, the search strategy(optimizer), and the performance estimation strategy (Elsken et al., 2018). Figure 1 shows how these components relate to each other.

2.1. Differentiable architecture search

The search space describes the way architectures are defined and what architectures can be discovered. A commonly used approach is a cell search space, in which small networks (cells) are learned which are then repeated in a macro-architecture. This discrete search space can also be made continuous by defining a one-shot architecture, a super-network consisting of all possible operations, and then applying continuous relaxation, adding a continuous meta-parameter for every possible option, as is done in DARTS (Liu et al., 2018).

The search strategy defines how the search space is explored to select candidate architectures, usually leveraging performance estimations of previous candidates to select more promising candidates. Common strategies include evolutionary algorithms (Real et al., 2017) and reinforcement learning methods (Baker et al., 2016). Gradient-based techniques can also be used: DARTS leverages weight sharing, optimizing both the meta-parameters

and the model weights at the same time using bilevel optimization, thus evaluating multiple architectures simultaneously. While this is very efficient, the downside is that many performance estimation strategies do not apply to DARTS.

2.2. Neural Architecture Optimization

A solution to this problem is offered by Neural Architecture Optimization (NAO), by Luo et al. (2018). It maps a neural network architecture x to an embedding space using an encoder-decoder model, as shown in Figure 2. The output of the encoder is a continuous feature vector e_x (i.e., an embedding), that captures the semantics of the architecture that served as input for the encoder. A performance predictor (NAO uses an MLP) then uses the embedding e_x to predict the architecture's accuracy. The output of the performance predictor is then maximized using gradient ascent to find e'_x, the embedding of an improved architecture x'. To get architecture x', the decoder maps e'_x back to the discrete space. This framework, using a performance predictor in a continuous search space, allows NAO to perform a gradient-based search in the embedding space quickly. An important limitation of NAO is that the performance predictor (MLP) first needs to be trained, which is a practical bottleneck. This paper, as described in detail in section 3, proposes a modification of NAO that uses a performance estimation strategy that makes the MLP redundant.

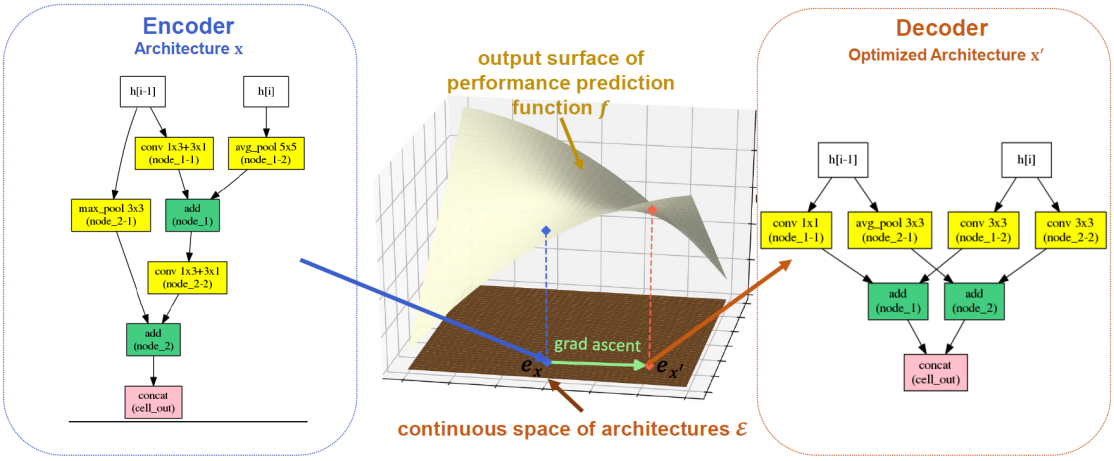

Figure 2: The NAO framework. Architecture x is mapped to a continuous embedding space by the encoder. Using its embedding e_x and predictor f, its performance is predicted. Then the performance of e_x is optimized into e'_x by maximizing the output of predictor f using gradient ascent. To get the improved architecture x', the decoder maps e'_x to the discrete space. From Luo et al. (2018).

2.3. Performance estimation strategies

Performance estimation strategies, or performance predictors, were developed since it is often infeasible in NAS to train every candidate architecture from scratch. NAO uses an

MLP, but many alternatives exist. These are often divided into the following categories: 1) lower fidelity estimates, 2) learning curve extrapolation, 3) weight inheritance, and 4) one-shot models/weight sharing. The first method reduces training times by training for fewer epochs, a subset of the data, downscaled models, etc. The second method aims to extrapolate model performance from a few epochs of training. In the third method, models are not trained from scratch, but initialized by inheriting weights from a parent model. The fourth category, used in DARTS, trains only a single large one-shot model, using the loss of the current one-shot model itself to optimize both the meta-parameters and model weights. Next to these methods, surrogate models can also be used to predict the performance of candidate pipelines. While NAO uses an MLP, we will leverage weighted kNN regression as a surrogate model.

3. Methods

EmProx uses an encoder-decoder meta model framework that maps candidate architectures to a continuous embedding space. EmProx estimates the performance of any new architecture based on the distance and accuracy of its nearest neighbors, previously evaluated candidate architectures, in the embedding space. The justification for this method is that in NAO training the MLP adds a significant time cost to the search, and would be redundant if we can predict the performance of architectures based on the embeddings in a simpler way. Our proposed method does exactly that: using the embeddings and the accuracy of other architectures the performance of candidates is predicted using kNN, which does not need any training, making the MLP redundant and thus lowering the time cost of the framework. Figure 3 shows the framework of EmProx and the remainder of this section describes this in more detail.

3.1. Learning the embedding

Neural network architectures can be represented as a directed acyclic graph. Such graphs can be represented as an adjacency matrix, as shown in Figure 4. Rows and columns represent operations such as convolutions and pooling layers and the values inside the adjacency matrix indicate the existence of a connection between operations in the graph. Consequently, the adjacency matrix can be shown as a sequence, such that a whole architecture can be encoded as a sequence.

This sequence, describing architecture a, serves as input for the framework. At training time, encoder EC maps architecture a to the continuous embedding space $\mathcal{E}$, denoted as $EC : \mathcal{A} \mapsto \mathcal{E}$. Then for all $a \in \mathcal{A}$ we have $e_a = EC(a)$, the embedding of architecture a. Given the high dimensionality of the embedding space, the embedding vectors are normalized. Then the decoder DC maps e_a back to its original architecture a, denoted as $DC : \mathcal{E} \mapsto \mathcal{A}$. During training, the negative log-likelihood loss between the original input and the output of the decoder is minimized. At inference time, to create embedding vectors for the query architectures, the sequences a describing their architectures are fed to encoder EC only.

Both the encoder and decoder consist of a single LSTM layer with a hidden dimension of 128, resulting in an embedding dimension of 128. This dimension was established empirically

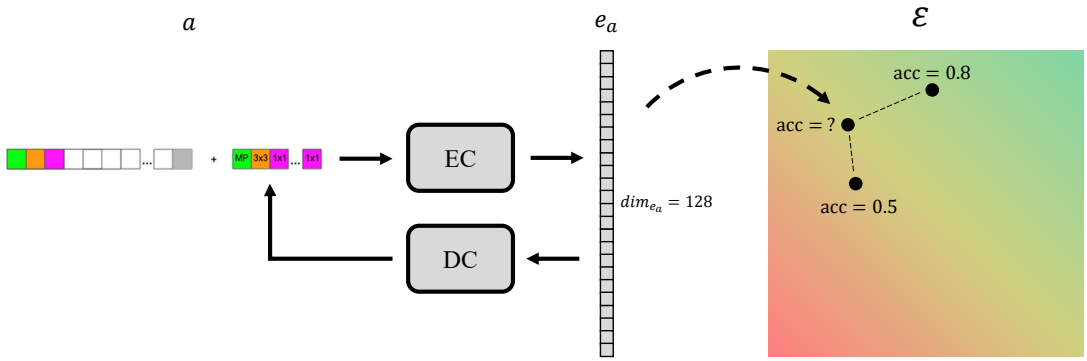

Figure 3: The EmProx framework. During training, architectures a encoded as a sequence are mapped to embedding space ϵ using encoder EC to create continuous architecture embeddings e_a. Consequently, they are mapped back to the discrete space, their original form a, by decoder DC to be able to learn informative embeddings. During inference, the k architectures with known validation accuracy that are the closest to each respective candidate architecture are used to estimate the accuracy of the candidate as weighted average using inverse distance weighting. In the image an example is shown with $k = 2$.

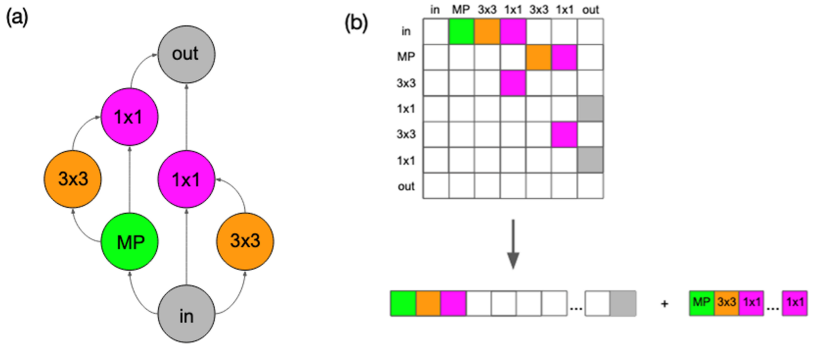

Figure 4: (a) the way layers and operations in a neural network are connected and (b) the representation as an adjacency matrix and vector.

in the experiments described in section 5. As in the original NAO paper, to better enable the decoder to reconstruct the input, an attention mechanism is used.

3.2. Performance estimation

We hypothesize that the encoder-decoder framework maps similar architectures (with similar performance) closely to each other in the embedding space, such that the performance of a new architecture can be inferred from architectures in its vicinity in the embedding space. This is capitalized on as follows. During inference, we have two sets of architectures: one set containing architectures with known accuracy $\mathcal{A}_{known}$ and one set of architectures or one shot model configurations with unknown accuracy $\mathcal{A}_{unknown}$ for which we want to predict the performance. Both sets are mapped to the embedding space by encoder EC such that we get two new sets. One set containing the embeddings of architectures with known accuracies $E_{\mathcal{A}_{known}} = \{EC(a) \mid a \in \mathcal{A}_{known}\}$ and one set of which we want to predict accuracies $E_{\mathcal{A}_{unknown}} = \{EC(a) \mid a \in \mathcal{A}_{unknown}\}$.

We then calculate the distance between all elements in these two sets $\{g(E_{\mathcal{A}_{known}} \times E_{\mathcal{A}_{unknown}})\}$. Here, g denotes the Euclidean distance function, which can be used given that the embedding vectors are normalized. Subsequently, we place the k closest neighboring architectures to each $a \in \mathcal{A}_{unknown}$ in set N_a. Here, k is a hyperparameter to be chosen later. Then for all $a \in \mathcal{A}_{unknown}$ and neighboring architectures $i = 1, ..., k$ in N_a we calculate the weight w_i based on inverse distance weighting. This is defined as $w_i = \frac{1}{g(e_a, e_i)}$. Lastly, the predicted accuracy $\hat{s}_a$ of architecture a is calculated as the weighted average of the accuracies s_i of neighbors $i = 1, ..., k$: $\hat{s}_a = \frac{\sum_{i=1}^{k} w_i(\mathbf{x}) s_i}{\sum_{i=1}^{k} w_i(\mathbf{x})}$.

Figure 5 shows a plot of the learned embeddings, where tSNE was used to reduce the dimensionality to 2. The colors, ranging from red (bad) to green (good), indicate the accuracy of the embedded architectures. As can be seen, by the red dotted clusters, several groups of badly performing architectures are grouped, indicating that the encoder-decoder framework indeed maps similar performing architectures closely to each other in the embedding space.

4. Implementation and Evaluation

To lower the computational barriers for NAS research, Ying et al. (2019) composed NAS-Bench 101, a dataset consisting of 423k unique neural network architectures and their performance evaluated on the CIFAR-10 dataset. This allows for benchmarking NAS algorithms without the necessity to train candidate models from scratch. Likewise, NAS-Bench-201 (Dong and Yang, 2020) operates on a cell-based search space, NAS-Bench-301 (Siems et al., 2020) on DARTS, and NAS-Bench-360 (Tu et al., 2021), on a more diverse set of tasks than CIFAR and Imagenet classification. The strategy proposed in this study is evaluated on NAS-Bench-201 to allow the fairest comparison with the strategies evaluated in White et al. (2021) using NASlib.

NASLib Ruchte et al. (2020) is a modular framework that implements many state-of-the-art NAS methods and building blocks like search spaces, search strategies, and performance

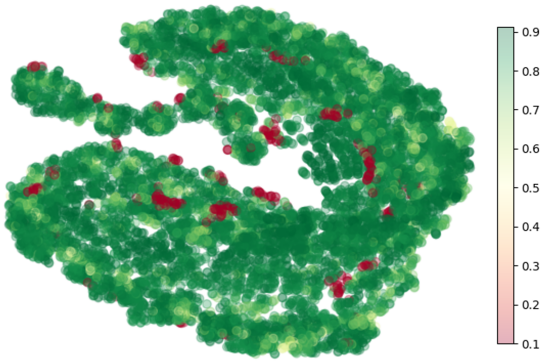

Figure 5: 2D plot of the learned architecture embeddings e_a after reducing the embedding dimesionality to 2 using tSNE. Colors indicate the accuracy of the respective embedded NAS-Bench-201 architectures on the CIFAR10 dataset. The plot shows groups of architectures with similar performance close to each other, which shows that performance of candidate architectures can be estimated from architectures that lie close by in embedding space

estimation strategies that can be used with only a few lines of code. White et al. (2021) added a total of 31 different neural architecture performance predictor methods to the NASLib framework, and compared them on NAS-Bench-201 with respect to initialization time, query time, and performance.

We evaluate how well our and other performance estimation strategies can predict the actual accuracy of a candidate model in terms of RMSE, MAE, and Pearson's correlation coefficient, Spearman's Rho, and Kendall Tau. Based on Pearson and Spearman, White et al. (2021) found that the best methods were Early Stopping (on validation accuracy), GCN (Wen et al., 2019), Jacobian Covariance (Mellor et al., 2020), LcSVR (Baker et al., 2017), SoTL-E (Ru et al., 2020), SemiNAS (Luo et al., 2020) and XGBoost (Chen and Guestrin, 2016). However, not all of these methods could be reasonably compared to ours. For instance, Early Stopping is not model-based and could hence not be compared in terms of fit time. Others, such as GCN, far exceeded our resource limitations, which made the comparison infeasible as well as less informative given our focus on efficiency and previously published benchmarks. Hence, to give a fair overview of competitors we chose to benchmark against NAO, SemiNAS, XGBoost and BANANAS (White et al., 2019) and an MLP.

To allow a direct comparison, we implemented our method as an addition to NASLib and evaluated it on NAS-Bench-201 architectures in the same manner as in White et al. (2021), as explained in section 5.

For training and testing, our evaluation framework randomly samples architectures from NAS-Bench-201 until a training set of 121 and a test set of 100 architectures are sampled (Random sampling could technically lead to data leakage, though this risk is deemed negligible due to the size of NAS-Bench and the impact of a single architecture being in both the train and test set). This means that 121 architectures, a value is taken from White et al. (2021), are used to learn the embeddings.

5. Empirical Evaluation

We implemented EmProx in NASLib and experiments were performed using the same framework as White et al. (2021) [2]. This ensured a fair comparison to other performance estimation strategies implemented in NASLib. In the experiments, the performance predictors of NAO, SemiNAS, XGBoost, BANANAS (White et al., 2019) and an MLP were tasked with estimating the performance of NAS-Bench-201 architectures on the CIFAR-10 dataset. In NAS-Bench, the architectures and their validation accuracies are stored. This allowed us to evaluate the performance of the predictors without having to train all architectures on the CIFAR-10 dataset from scratch. As a result, no GPU was required and all experiments have been performed on a single Intel Xeon E5-2698v4 CPU.

2*Method	Regression metrics		Correlation coefficients			Time metrics	
	MAE	RMSE	Pearson	Spearman	Kendall	Fit time(s)	Query time(ms)
EmProx ($k = 10$)	4.4027 ± 1.0269	10.7163 ± 2.9690	0.4771 ± 0.1497	0.7304 ± 0.0645	0.5453 ± 0.0537	6.4498 ± 0.9846	0.9 ± 0.8
EmProx ($k = 60$)	4.5264 ± 0.9625	10.7953 ± 3.1221	0.5044 ± 0.0963	0.7332 ± 0.0865	0.5468 ± 0.0718	7.2310 ± 1.0598	3.2 ± 0.4
NAO	4.7336 ± 1.0260	10.9394 ± 2.8279	0.4512 ± 0.1335	0.7131 ± 0.0657	0.5279 ± 0.0608	54.1674 ± 0.3493	2.6 ± 0.2
SemiNAS	4.0283 ± 0.8023	10.1222 ± 2.2704	0.5307 ± 0.1604	0.7677 ± 0.0568	0.5822 ± 0.0564	152.2268 ± 42.5809	1.2 ± 0.7
XGB	5.3989 ± 1.3110	12.2955 ± 3.5906	0.4008 ± 0.2224	0.6466 ± 0.0597	0.4719 ± 0.0476	31.6526 ± 3.9272	0.4 ± 0.1
BANANAS	7.3007 ± 1.1988	12.2760 ± 2.0153	0.3793 ± 0.1937	0.4169 ± 0.0959	0.2910 ± 0.0653	507.3936 ± 13.87	0.2 ± 0.1
MLP	6.8584 ± 1.6152	11.3592 ± 2.2580	0.4417 ± 0.1142	0.5298 ± 0.0996	0.3759 ± 0.0777	471.7508 ± 6.2376	<0.1 ± <0.1

Table 1: Experiment results on the CIFAR10 dataset with NAS-Bench-201 architectures, averaged over 20 trials. Above are the results of the method proposed in this study, below are several methods evaluated in White et al. (2021). Underlined are the best results of EmProx and the best results among the other predictors for ease of comparison. As can be seen, EmProx scores competitively regarding regression and correlation coefficients and scores much better regarding fit time.

As part of the experiments, a grid search was performed to find the dimension of the hidden layer (i.e. embedding dimension) and EmProx hyperparameter k. Other models and training parameters were left unchanged and taken from White et al. (2021) and the paper of NAO by Luo et al. (2018).

The results of the performance predictors were expressed in terms of regression metrics MAE and RMSE between the predicted candidate architecture accuracy and the validation accuracy from NAS-Bench. Additionally, it has to be noted that in many NAS algorithms the exact predicted performance is not of importance, rather than the rank of a certain architecture among other candidates, since only the most promising architectures are trained. This is incorporated in the experiments by including correlation-based performance measures Pearson, Spearman, and Kendall's Tau on the predicted and validation accuracies. Furthermore, since ultimately the goal of performance predictors is to speed up the search, fit time and query time are included as measures. For EmProx the fit time includes the learning of the embeddings and the query time the creation of the embedding for a query architecture and the performance estimation. The hyperparameter search was conducted

2. Available at https://github.com/automl/naslib

to explore the sensitivity and performance of the method towards its hyperparameters and is hence not included in the fit time.

From the grid search, it was found that a hidden layer dimension (i.e. embedding dimension) of 32 yielded the best results. Regarding the number of neighbors k, it was found that $k = 10$ led to the best results in terms of the MAE, RMSE, and fit time. For the rank and correlation coefficients, $k = 60$ reached slightly better results. The result on the CIFAR10 dataset using NAS-Bench-201 architectures can be found in Table 1, alongside the results of high-performing performance predictors from White et al. (2021) for comparison. All results have been averaged out over 20 trials.

Compared with the performance predictor of NAO, EmProx outperforms NAO on all metrics. This was to be expected for the fit time since EmProx does not need to train the MLP that NAO uses for the performance prediction. The fact that it also outperforms NAO in terms of the regression metrics and correlation coefficients counts as empirical evidence that the method proposed in this study is viable. Regarding the regression and correlation coefficients, SemiNAS yields better results. However, this comes at the cost of the fit time; EmProx is over 20 times faster than SemiNAS. Compared to the second-fastest method, XGBoost, Emprox is still around 5 times faster and performs better on both regression metrics as well as correlation coefficients.

To further validate the performance of EmProx, additional experiments have been conducted. Table 2 shows the results of performance prediction of NAS-Bench-201 architectures on the CIFAR100 dataset. This dataset is more challenging than CIFAR10. On the regression metrics and correlation coefficients, our method again scores slightly worse than the best predictor from the previous experiments, SemiNAS. However, while on this complicated dataset the fit time of SemiNAS is becoming infeasibly high, EmProx is still outstandingly fast.

Another experiment has been conducted on the DARTS search space. Averaged over twenty trials, EmProx had a RMSE of 0.6652 ± 0.0694 and a Pearson correlation of 0.4537 ± 0.0822 with a fit time of 146.1185 ± 6.9496. These results could not be compared to SemiNAS, since SemiNAS exceeded our resource limitation of 24 hours of CPU time.

These experiments corroborate the findings that our method scores competitively on regression metrics and correlation coefficients with an immensely lower fit time.

2*Method	Regression metrics		Correlation coefficients			Time metrics (s)	
	MAE	RMSE	Pearson	Spearman	Kendall	Fit time	Query time
EmProx ($k = 60$)	5.1859 ± 0.8155	9.914± 2.3226	0.6212 ± 0.0792	0.7772 ± 0.0490	0.5877 ± 0.0427	16.149 ± 0.676	0.004 ± 0.001
SemiNAS	4.5925 ± 0.5693	9.0315 ± 1.5319	0.6469 ± 0.1034	0.7990 ± 0.0428	0.6124 ± 0.0418	1309.134 ± 53.124	0.010 ± 0.004

Table 2: Experiment results on the CIFAR100 dataset with NAS-Bench-201 architectures, averaged over 20 trials. Compared to the best performer in earlier experiments, SemiNAS, our method again scores slightly worse on regression metrics and correlation coefficients, but the fit time is order of magnitude.

6. Conclusion and Future Work

In this study we proposed Embedding proximity score (EmProx), a cheaper and faster method to do performance estimation during neural architecture search. In the empirical results EmProx was approximately nine times faster than NAO and up to eighty times faster than alternatives, while yielding similarly good predictions according to various regression metrics and correlation coefficients. It can be interesting to integrate EmProx as a part of full NAS pipelines in future to check for overall performance improvement. We would like to compare EmProx with other meta methods for performance estimation. In future we would like to extend this approach for search process as well.

References

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016. URL http://arxiv.org/abs/1611.02167.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Practical neural network performance prediction for early stopping. *CoRR*, abs/1705.10823, 2017. URL http://arxiv.org/abs/1705.10823.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016. URL http://arxiv.org/abs/1603.02754.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *CoRR*, abs/2001.00326, 2020. URL http://arxiv.org/abs/2001.00326.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A Survey. *Journal of Machine Learning Research*, 20, aug 2018. ISSN 15337928. URL https://arxiv.org/abs/1808.05377v3.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. *7th International Conference on Learning Representations, ICLR 2019*, jun 2018. URL https://arxiv.org/abs/1806.09055v2.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, volume 2018-Decem, pages 7816–7827. Neural information processing systems foundation, aug 2018. URL https://arxiv.org/abs/1808.07233v5.

Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *CoRR*, abs/2002.10389, 2020. URL https://arxiv.org/abs/2002.10389.

Joseph Mellor, Jack Turner, Amos J. Storkey, and Elliot J. Crowley. Neural architecture search without training. *CoRR*, abs/2006.04647, 2020. URL https://arxiv.org/abs/2006.04647.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc V. Le, and Alex Kurakin. Large-scale evolution of image classifiers. *CoRR*, abs/1703.01041, 2017. URL http://arxiv.org/abs/1703.01041.

Binxin Ru, Clare Lyle, Lisa Schut, Miroslav Fil, Mark van der Wilk, and Yarin Gal. Speedy Performance Estimation for Neural Architecture Search. jun 2020. doi: 10.48550/arxiv.2006.04492. URL https://arxiv.org/abs/2006.04492v1http://arxiv.org/abs/2006.04492.

Michael Ruchte, Arber Zela, Julien Siems, Josif Grabocka, and Frank Hutter. Naslib: A modular and flexible neural architecture search library. https://github.com/automl/NASLib, 2020.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *CoRR*, abs/2008.09777, 2020. URL https://arxiv.org/abs/2008.09777.

Renbo Tu, Mikhail Khodak, Nicholas Roberts, and Ameet Talwalkar. NAS-Bench-360: Benchmarking Diverse Tasks for Neural Architecture Search. oct 2021. URL http://arxiv.org/abs/2110.05668.

Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. *CoRR*, abs/1912.00848, 2019. URL http://arxiv.org/abs/1912.00848.

Colin White, Willie Neiswanger, and Yash Savani. BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. oct 2019. doi: 10.48550/arxiv.1910.11858. URL https://arxiv.org/abs/1910.11858v3http://arxiv.org/abs/1910.11858.

Colin White, Arber Zela, Binxin Ru, Yang Liu, Frank Hutter, and Abacus Ai. How Powerful are Performance Predictors in Neural Architecture Search? apr 2021. URL https://arxiv.org/abs/2104.01177v2.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *CoRR*, abs/1902.09635, 2019. URL http://arxiv.org/abs/1902.09635.