Synthetic Petri Dish: A Novel Surrogate Model for Rapid Architecture Search

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural Architecture Search (NAS) explores a large space of architectural motifs – a compute-intensive process that often involves ground-truth evaluation of each motif by instantiating it within a large network, and training and evaluating the network with thousands or more data samples. Inspired by how biological motifs such as cells are sometimes extracted from their natural environment and studied in an artificial Petri dish setting, this paper proposes the *Synthetic Petri Dish* model for evaluating architectural motifs. In the Synthetic Petri Dish, architectural motifs are instantiated in very small networks and evaluated using very few *learned* synthetic data samples (to effectively approximate performance in the full problem). The relative performance of motifs in the Synthetic Petri Dish can substitute for their ground-truth performance, thus accelerating the most expensive step of NAS. Unlike other neural network-based prediction models that parse the structure of the motif to *estimate* its performance, the Synthetic Petri Dish predicts motif performance by training *the actual motif* in an artificial setting, thus deriving predictions from its true intrinsic properties. Experiments in this paper demonstrate that the Synthetic Petri Dish can therefore predict the performance of new motifs with significantly higher accuracy, especially when insufficient ground truth data is available. Our hope is that this work can inspire a new research direction in studying the performance of extracted components of models in a synthetic diagnostic setting optimized to provide informative evaluations.

1 Introduction

The architecture of deep neural networks (NNs) is critical to their performance. This fact motivates neural architecture search (NAS), wherein the choice of architecture is often framed as an automated search for effective *motifs*, i.e. the design of a repeating recurrent cell or activation function that is repeated often in a larger NN blueprint. However, evaluating a candidate architecture's ground-truth performance in a task of interest depends upon training the architecture to convergence. Complicating efficient search, the performance of an architectural motif nearly always benefits from increased computation (i.e. larger NNs trained with more data). The implication is that the best architectures often require training near the bounds of what computational resources are available, rendering naive NAS (i.e. where each candidate architecture is trained to convergence) exorbitantly expensive.

To reduce the cost of NAS, methods often exploit heuristic *surrogates* of true performance. For example, motif performance can be evaluated after a few epochs of training or with scaled-down architectural blueprints, which is often still expensive (because maintaining reasonable fidelity between ground-truth and surrogate performance precludes aggressive scaling-down of training). Another approach learns models of the search space (e.g. Gaussian processes models used within Bayesian optimization), which improve as more ground-truth models are trained, but cannot generalize

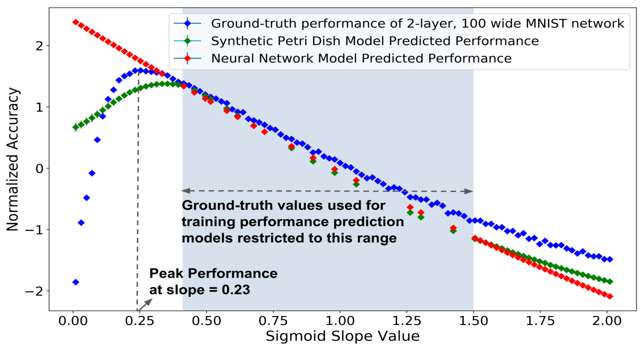


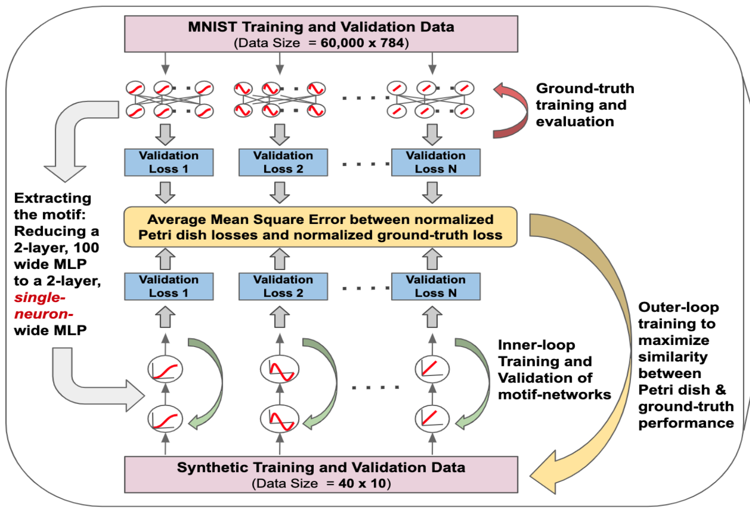
Figure 1: **Predicting the Optimal Slope of the Sigmoid Activation.** Each blue diamond depicts the normalized validation accuracy (ground-truth) of a 2-layer, 100-wide feed-forward network with a unique sigmoid slope value (mean of 20 runs). The validation accuracy peaks at a slope of 0.23. Both the Synthetic Petri Dish and a neural network surrogate model that predicts performance as a function of sigmoid slope are trained on a limited set of ground-truth points, restricted to the blue-shaded region to the right of the peak. The normalized performance predictions for Synthetic Petri Dish are shown with green diamonds and those for the NN surrogate model are shown as red diamonds. The plot shows that the NN model predictions overfits the training data. In contrast, because the Synthetic Petri Dish conducts experiments with small neural networks with these sigmoid slope values it is more accurate at inferring both that there is a peak and its approximate location.

well beyond the examples seen. This paper explores whether the computational efficiency of NAS can be improved by creating a new kind of surrogate, one that can benefit from miniaturized training and still generalize beyond the observed distribution of ground-truth evaluations. To do so, we take inspiration from an idea in biology, bringing to machine learning the application of a *Synthetic Petri Dish* microcosm that aims to identify high-performing architectural motifs.

The overall motivation behind "in vitro" (test-tube) experiments in biology is to investigate in a simpler and controlled environment the key factors that explain a phenomenon of interest in a messier and more complex system. For example, to understand causes of atypical mental development, scientists extract individual neuronal cells taken from brains of those demonstrating typical and atypical behavior and study them in a Petri dish [1]. The approach proposed in this paper attempts to algorithmically recreate this kind of scientific process for the purpose of finding better neural network motifs. The main insight is that biological Petri dish experiments often leverage both (1) key aspects of a system's dynamics (e.g. the behavior of a single cell taken from a larger organism) and (2) a human-designed intervention (e.g. a measure of a test imposed on the test-tube). In an analogy to NAS, (1) the dynamics of learning through backpropagation are likely important to understanding the potential of a new architectural motif, and (2) compact synthetic datasets can illuminate an architecture's response to learning. That is, we can use machine learning to *learn* data such that training an architectural motif on the learned data results in performance indicative of the motif's ground-truth performance.

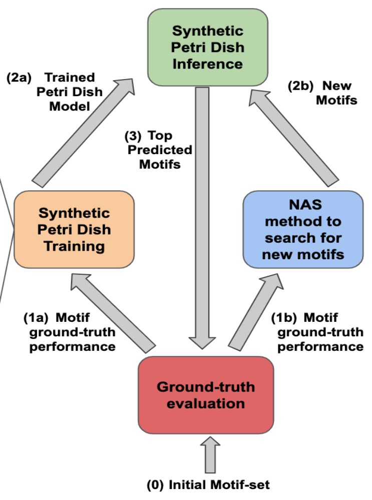
In the proposed approach, motifs are extracted from their *ground-truth evaluation* setting (i.e. from large-scale NNs trained on the full dataset of the underlying domain of interest, e.g. MNIST), instantiated into very small networks (called *motif-networks*), and evaluated on learned synthetic data samples. These synthetic data samples are trained such that the *performance ordering* of motifs in this Petri dish setting (i.e. a miniaturized network trained on a few synthetic data samples) matches their ground-truth performance ordering. Because the *relative* performance of motifs is sufficient to distinguish good motifs from bad ones, the Petri dish evaluations of motifs can be a surrogate for ground-truth evaluations in NAS. Training the Synthetic Petri Dish is also computationally inexpensive, requiring only a few ground-truth evaluations, and once trained it enables extremely rapid evaluations of new motifs.

A key motivating hypothesis is that because the Synthetic Petri Dish evaluates the motif by actually *using* it in a simple experiment (e.g. training it with SGD and then evaluating it), its predictions can generalize better than other neural network (NN) based models that predict motif performance based on only observing the motif's structure and resulting performance [2, 3]. For example, consider the demonstration problem of predicting the ground-truth performance of a two-layer feedforward MNIST network with sigmoidal non-linearity. The blue points in Figure 1 shows how the ground-truth performance of the MNIST network varies when the slope of its sigmoid activations (the term c in the sigmoid formula $1/(1 + e^{-cx})$) is varied in the range of $0.01 - 2.01$. The MNIST network performance peaks near a slope-value of 0.23. Similarly to the NN-based model previously developed



MNIST Training and Validation Data
(Data Size = 60,000 x 784)
Ground-truth training and evaluation
Validation Loss 1
Validation Loss 2
Validation Loss N
Extracting the motif: Reducing a 2-layer, 100 wide MLP to a 2-layer, single-neuron-wide MLP
Average Mean Square Error between normalized Petri dish losses and normalized ground-truth loss
Outer-loop training to maximize similarity between Petri dish & ground-truth performance
Inner-loop Training and Validation of motif-networks
Synthetic Training and Validation Data
(Data Size = 40 x 10)

(a) Synthetic Petri Dish Training Procedure



Synthetic Petri Dish Inference
(2a) Trained Petri Dish Model
(2b) New Motifs
(3) Top Predicted Motifs
Synthetic Petri Dish Training
NAS method to search for new motifs
(1a) Motif ground-truth performance
(1b) Motif ground-truth performance
Ground-truth evaluation
(0) Initial Motif-set

(b) Combining Architecture Search with the Petri Dish

Figure 2: (a) **Synthetic Petri Dish Training.** The left figure illustrates the inner and outer-loop training procedure. The motifs (in this example, activation functions) are extracted from the full network (e.g a 2-layer, 100 wide MLP) and instantiated in separate, much smaller motif-networks (e.g. a two-layer, single-neuron MLP). The motif-networks are trained in the inner-loop with the synthetic training data and evaluated using synthetic validation data. In the outer-loop, an average MSE loss is computed between the normalized Petri dish validation losses and the corresponding normalized ground-truth losses. Synthetic training and validation data are optimized by taking gradient steps w.r.t the outer-loop loss. (b) **Combining Architecture Search with Petri Dish.**

in Liu et al. [2], Luo et al. [3], one can try to train a neural network that predicts the performance of the corresponding MNIST network given the sigmoid slope value as input (Section 3.1 provides full details). When training points (tuples of sigmoid slope value and its corresponding MNIST network performance) are restricted to an area to the right of the peak (Figure 1, blue-shaded region), the NN-based prediction model (Figure 1, red diamonds) generalizes poorly to the test points on the left side of the peak ($c < 0.23$). However, unlike such a conventional prediction model, the prediction of the Synthetic Petri Dish generalizes to test points left of the peak. That occurs because the Synthetic Petri Dish trains and evaluates the *actual candidate motifs*, rather than just making predictions about their performance based on data from past trials.

Beyond this explanatory experiment, the promise of the Synthetic Petri Dish is further demonstrated on a challenging and compute-intensive language modelling task that serves as a popular NAS benchmark. Interestingly, these results suggest that it is indeed possible to extract a motif from a larger setting and create a controlled setting (through learning synthetic data) where the instrumental factor in the performance of the motif can be isolated and tested quickly, just as scientists use Petri dishes to test specific hypothesis to isolate and understand causal factors in biological systems.

2 Methods

Recall that the aim of the Synthetic Petri Dish is to create a microcosm training environment such that the performance of a small-scale motif trained within it well-predicts performance of the fully-expanded motif in the ground-truth evaluation. First, a few initial ground-truth evaluations of motifs are needed to create training data for the Petri dish. In particular, consider N motifs for which ground-truth validation loss values ($\mathcal{L}^i_{true}$, where $i \in 1, 2, ...N$) have already been pre-computed by training each motif in the ground-truth setting. The next section details how these initial evaluations are leveraged to train the Synthetic Petri Dish.

2.1 Training the Synthetic Petri Dish

To train the Petri Dish first requires extracting the N motifs from their ground-truth setting and instantiating each of them in miniature as separate *motif-networks*. For the experiments in this paper, the ground-truth network and the motif-network have the same overall blueprint and differ only in the width of their layers. For example, Figure 2a shows a ground-truth network's size reduced from a 2-layer, 100-wide MLP to a motif-network that is a 2-layer MLP with a *single neuron* per layer.

Given such a collection of extracted motif-networks, a small number of *synthetic* training and validation data samples are then learned that can respectively be used to train and evaluate the motif-networks. The learning objective is that the validation loss of motifs trained in the Petri dish resemble the validation loss of the motif's ground-truth evaluation ($\mathcal{L}^i_{true}$). Note that this training process requires two nested optimization loops: an inner-loop that trains and evaluates the motif-networks on the synthetic data and an outer-loop that trains the synthetic data itself.

Initializing the Synthetic Petri Dish: Before training the Petri dish, the motif-networks and synthetic data must be initialized. Once the motifs have been extracted into separate motif-networks, each motif-network is assigned *the same* initial random weights (θ_{init}). This constraint reduces confounding factors by ensuring that the motif-networks differ from each other *only* in their instantiated motifs. At the start of Synthetic Petri Dish training, synthetic training data ($S^{train} = (x^{train}, y^{train})$) and validation data samples ($S^{valid} = (x^{valid}, y^{valid})$) are randomly initialized. Note that these learned training and validation data can play distinct and complementary roles, e.g. the validation data can learn to test out-of-distribution generalization from a learned training set. Empirically, setting the training and validation data to be the same initially (i.e. $S^{train} = S^{valid}$) benefited optimization at the beginning of outer-loop training; over iterations of outer-loop training, the synthetic training and validation data then diverge. The size of the motif-network and the number of synthetic data samples are chosen through the hyperparameter selection procedure described in Appendix A.3.

Inner-loop training: The inner optimization loop is where the performance of motif-networks is evaluated by training each such network independently with synthetic data. This training reveals a sense of the quality of the motifs themselves.

In each inner-loop, the motif-networks are *independently trained* with SGD using the synthetic training data (S^{train}). The motif-networks take synthetic training inputs (x^{train}) and produce their respective output predictions ($\hat{y}^{train}$). For each motif-network, a binary cross-entropy (BCE) loss is computed between the output predictions ($\hat{y}^{train}$) and the synthetic training labels (y^{train}). Because the Petri dish is an artificial setting, the choice of BCE as the inner-loop loss ($\mathcal{L}_{inner}$) is independent of the actual domain loss (used for ground-truth training), and other losses like regression loss could instead be used. The gradients of the BCE loss w.r.t. the motif-network weights inform weight updates (as in regular SGD).

$$\theta^i_{t+1} = \theta^i_t - \alpha \nabla \mathcal{L}^i_{inner\_train}(S^{train}, \theta^i_t) \quad i \in 1, 2, .., N \tag{1}$$

where α is the inner-loop learning rate and $\theta^i_0 = \theta_{init}$. Inner-loop training proceeds until individual BCE losses converge. Once trained, each motif-network is independently evaluated using the synthetic validation data (S^{valid}) to obtain individual validation loss values ($\mathcal{L}^i_{inner\_valid}$). These inner-loop validation losses then enable calculating an outer-loop loss to optimize the synthetic data, which is described next.

Outer-loop training: Recall that an initial sampling of candidate motifs evaluated in the ground-truth setting serve as a training signal for crafting the Petri dish's synthetic data. That is, in the outer loop, synthetic training data is optimized to encourage motif-networks trained upon it to become accurate surrogates for the performance of full networks built with that motif evaluated in the ground-truth setting. The idea is that training motif-networks on the right (small) set of synthetic training data can potentially isolate the key properties of candidate motifs that makes them effective.

To frame the outer-loop loss function, what is desired is for the validation loss of the motif-network to induce the same *relative ordering* as the validation loss of the ground-truth networks; such relative ordering is all that is needed to decide which new motif is likely to be best. One way to design such an outer-loop loss with this property is to penalize differences between normalized loss values in the Petri dish and ground-truth setting. To this end, the motif-network (inner-loop) loss values and their respective ground-truth loss values are first independently normalized to have zero-mean and unit-variance. Then, for each motif, a mean squared error (MSE) loss is computed between the normalized inner-loop validation loss ($\hat{\mathcal{L}}^i_{inner\_valid}$) and the normalized ground-truth validation loss ($\hat{\mathcal{L}}^i_{true}$). The MSE loss is averaged over all the motifs and used to compute a gradient step to improve the synthetic training and validation data.

$$\mathcal{L}_{outer} = \frac{1}{N} \sum_{i=1}^{N} (\hat{\mathcal{L}}^i_{inner\_valid} - \hat{\mathcal{L}}^i_{true})^2 \tag{2}$$

$$S_{t+1}^{train} = S_t^{train} - \beta \nabla \mathcal{L}_{outer} \quad \text{and} \quad S_{t+1}^{valid} = S_t^{valid} - \beta \nabla \mathcal{L}_{outer} \tag{3}$$

where β is the outer-loop learning rate. For simplicity, only the synthetic training (x^{train}) and validation (x^{valid}) inputs are learned and the corresponding labels (y^{train}, y^{valid}) are kept fixed to their initial random values throughout training. Minimizing the outer-loop MSE loss ($\mathcal{L}_{outer}$) modifies the synthetic training and validation inputs to maximize the similarity between the motif-networks' performance ordering and motifs' ground-truth ordering.

After each outer-loop training step, the motif-networks are reset to their original initial weights (θ_{init}) and the inner-loop training and evaluation procedure (equation 1) is carried out again. The outer-loop training proceeds until the MSE loss converges, resulting in optimized synthetic data.

2.2 Predicting Performance with the Trained Petri Dish

The Petri Dish training procedure described so far results in synthetic training and validation data optimized to sort motif-networks similarly to the ground-truth setting. This section describes how the trained Petri dish can predict the relative performance of unseen motifs, which we call the *Synthetic Petri Dish inference* procedure. In this procedure, new motifs are instantiated in their individual motif-networks, and the motif-networks are trained and evaluated using the optimized synthetic data (with the same hyperparameter settings as in the inner-loop training and evaluation). The relative inner-loop validation loss for the motif-networks then serves as a surrogate for the motifs' relative ground-truth validation loss. Such Petri dish inference is computationally inexpensive because it involves training and evaluation of very small motif-networks with very few synthetic examples.

Interestingly, the cheap-to-evaluate surrogate performance prediction given by the trained Petri dish is complementary to most NAS methods that search for motifs, meaning that they can easily be combined. Algorithm 1 in Appendix A.2 shows one possible hybridization of Petri dish and NAS.

3 Experiments

3.1 Searching for the Optimal Slope for Sigmoidal Activation Functions

Preliminary experiments demonstrated that when a 2-layer, 100-wide feed-forward network with sigmoidal activation functions is trained on MNIST data, its validation accuracy (holding all else constant) depends on the slope of the sigmoid (blue curve in Figure 1).

Both the Synthetic Petri Dish and the NN-based surrogate model are trained using 30 ground-truth points that are randomly selected from a restricted interval of sigmoid slope values (the blue-shaded region in Figure 1). The remaining ground-truth points (outside the blue-shaded region) are used only for testing. The NN-based surrogate control is a 2-layer, 10-wide MLP that takes the sigmoid value as input and predicts the corresponding MNIST network validation accuracy as its output.

For training the Synthetic Petri Dish model, each training motif (i.e. sigmoid with a unique slope value) is extracted from the MNIST network and instantiated in a 2-layer, *single-neuron*-wide motif-network (θ_{init}). The setup is similar to the one shown in Figure 2a). The motif-networks are trained in the inner-loop for 200 SGD steps and subsequently their performance on the synthetic validation data ($\mathcal{L}^{i}_{inner\_valid}$) is used to compute outer-loop MSE loss (as described in section 2.1). A total of 20 outer-loop training steps are performed (see Appendix A.4 for details).

Results demonstrate that the NN-based model overfits the training points while in contrast, Petri Dish accurately infers the location of the peak (red and green-curves respectively in Figure 1).

3.2 Architecture Search for Recurrent Cells

The previous experiment demonstrated that a Synthetic Petri Dish model trained with limited ground-truth data can successfully generalize to unseen out-of-distribution motifs. This next experiment tests whether the Synthetic Petri Dish can be applied to a more realistic and challenging setting, that of NAS for a NN language model that is applied to the Penn Tree Bank (PTB) dataset – a popular language modeling and NAS benchmark [4]. In this experiment, the architectural motif is the design of a recurrent cell. The recurrent cell search space and its ground-truth evaluation setup is the same as in NAO [3]. This NAS problem is challenging because the search space is expansive and few solutions perform well [5]. Each cell in the search space is composed of 12 layers (each with the same

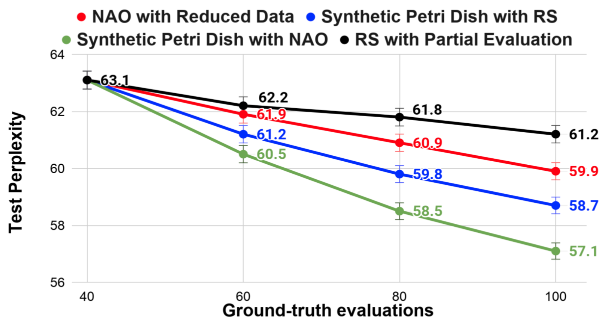


Figure 3: **RNN Cell Search for PTB:** This graph plots the test perplexity (mean of five runs) of the best found cell for four NAS methods across variable numbers of NAS iterations. All the methods are warmed up at the beginning (Step 0 in Figure 2b) with 40 ground-truth evaluations – notice the top-left point with best test perplexity of 63.1. For **Synthetic Petri Dish with RS** (blue curve) and **Synthetic Petri Dish with NAO** (green curve), the top 20 motifs with the best predicted performance are selected for ground-truth evaluations in each NAS iteration. Original NAO [3] (not shown here) requires 1,000 ground-truth evaluations to achieve a test perplexity of 56.0. **NAO with Reduced Data** (red-curve) shows the results obtained by running original NAO, but with only 100 ground-truth evaluations (the same number the Synthetic Petri Dish variants get). With such limited data, **Synthetic Petri Dish with NAO** outperforms other NAS methods and achieves a test perplexity of 57.1.

layer width) that are connected in a unique pattern. An input embedding layer and an output soft-max layer is added to the cell (each of layer width 850) to obtain a full network (27 Million parameters) for ground-truth evaluation. Each ground-truth evaluation requires training on PTB data-set for 600 epochs and takes 10 hours on a Nvidia 1080Ti.

NAO is one of the state-of-the-art NAS methods for this problem and is therefore used as a baseline in this experiment (called here *original NAO*). In the published results [3], *original NAO* requires 1,000 ground-truth evaluations (300 GPU days) over three successive *NAS iterations* to discover a cell with test perplexity of 56.0. These are good results, but the compute cost even to reproduce them is prohibitively large for many researchers. Because the Synthetic Petri Dish offers a potential low-compute option, in this experiment, different NAS methods are compared instead in a setting where only limited ground-truth evaluation data is available (≤ 100 samples), giving a sense of how far different methods can get with more reasonable compute resources.

Each *NAS iteration* can be accelerated if the number of costly ground-truth evaluations is reduced by instead cheaply evaluating the majority of candidate motifs (i.e. new cells) in the Petri dish. For the purpose of training the Synthetic Petri Dish, each cell is extracted from its ground-truth setting (850 neurons per layer) and is instantiated in a motif-network with three neurons per layer (its internal cell connectivity, including its depth, remains unchanged). Thus, the ground-truth network that has 27 million parameters is reduced to a motif-network with only 140 parameters. To train motif-networks, synthetic training and validation data each of size $20 \times 10 \times 10$ *(batch size$\times$time steps$\times$motif-network input size)* is learned (thus replacing the 923k training and 73k validation words of PTB). The Petri dish training and inference procedure is very similar to the one described in Experiment 3.1, and it adds negligible compute cost (2 extra hours for training, and a few minutes for inference on a CPU).

Following the steps outlined in algorithm 1 and Figure 2b, the Petri dish surrogate can be combined with two existing NAS methods: (1) Random Search (RS) or (2) NAO itself, resulting in two new methods called Synthetic Petri Dish-RS and Synthetic Petri Dish-NAO. Also, the Random Search NAS method can be combined with partial evaluations resulting in another baseline (Appendix A.5).

The test perplexity of the best found motif at the end of each NAS iteration is plotted in Figure 3 – the blue curve depicts the result for Synthetic Petri Dish-RS and green depicts the result for Synthetic Petri Dish-NAO. For a fair comparison, original NAO is re-run in this limited ground-truth setting and the resulting performance is depicted by the red-curve in Figure 3. The results show that Synthetic Petri Dish-NAO outperforms both Synthetic Petri Dish-RS and NAO when keeping the amount of ground-truth data points the same, suggesting that the Synthetic Petri Dish and NAO complement each other well. The hybridization of Synthetic Petri Dish and NAO finds a cell that is competitive in its performance (test perplexity 57.1) with original NAO (56.0), using only $1/10^{th}$ of original NAO's compute (and exceeds the performance of original NAO when both are given equivalent compute).

4 Discussion and Conclusions

Results from the sigmoid slope search and the NAS for recurrent cells experiment indicate that Synthetic Petri Dish can capture a set of priors that a naive black box performance predictor would often miss (see Figure A.4.2). Teasing out such underlying causal factors of performance is a novel research direction that may ultimately teach us new lessons on architecture by exposing the most important dimensions of variation through a principled empirical process.

References

[1] D. Adhya, E. Annuario, M. A. Lancaster, J. Price, S. Baron-Cohen, and D. P. Srivastava. Understanding the role of steroids in typical and atypical brain development: Advantages of using a "brain in a dish" approach. *Journal of Neuroendocrinology*, 30(2):e12547, 2018. doi: 10.1111/jne.12547. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/jne.12547`.

[2] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[3] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pages 7816–7827, 2018.

[4] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2), 1993.

[5] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[6] P. Langley. Crafting papers on machine learning. In Pat Langley, editor, *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

[7] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. 2017. URL `https://arxiv.org/abs/1611.01578`.

[8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *CoRR*, abs/1808.05377, 2018.

[9] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

[10] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 497–504, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3071229. URL `http://doi.acm.org/10.1145/3071178.3071229`.

[11] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. In Robert Kozma, Cesare Alippi, Yoonsuck Choe, and Francesco Carlo Morabito, editors, *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam: Elsevier, 2018.

[12] 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, pages 4780–4789, 2019.

[13] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019.

[14] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. 2016. URL `https://arxiv.org/pdf/1611.02167v2.pdf`.

[15] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[16] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *ICLR*, 2017.

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

[18] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2016–2025. Curran Associates, Inc., 2018.

[19] Shengcao Cao, Xiaofang Wang, and Kris M. Kitani. Learnable embedding space for efficient neural architecture compression. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=S1xLN3C9YX`.

[20] Hanzhang Hu, John Langford, Rich Caruana, Saurajit Mukherjee, Eric Horvitz, and Debadeepta Dey. Efficient forward architecture search. *Neural Information Processing Systems*, 2019.

[21] Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data, 2020. URL `https://openreview.net/forum?id=HJg_ECEKDr`.

[22] Prajit Ramachandran, Barret Zoph, and Quoc Le. Searching for activation functions. 2018. URL `https://arxiv.org/pdf/1710.05941.pdf`.