

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

Aditya Rawal¹ Joel Lehman¹ Felipe Petroski Such¹ Jeff Clune^{*2} Kenneth O. Stanley^{*1}

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 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

^{*}Co-senior authors ¹Uber AI Labs ²Open AI. Work done at Uber AI Labs. Correspondence to: Aditya Rawal <aditya.rawal@uber.com>.

study them in a Petri dish (Adhya et al., 2018). 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 (Liu et al., 2018a; Luo et al., 2018). 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 in Liu et al. (2018a); Luo et al. (2018), 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 4.1 provides full details). When training points (tuples of sigmoid slope value and its corresponding MNIST

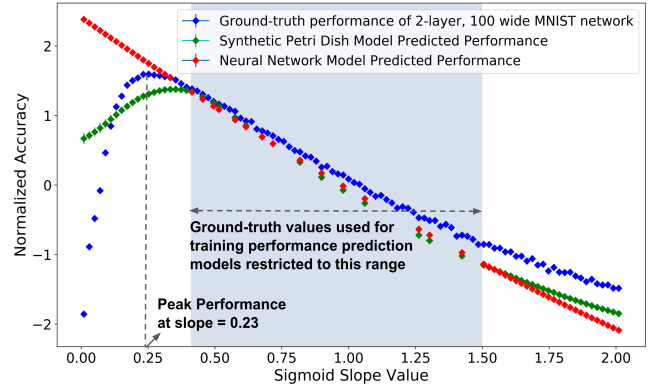


Figure 1. Predicting the Optimal Slope of the Sigmoid Activation. Each blue diamond depicts the normalized validation accuracy (ground-truth) of a 2-layer, 100-wide feed-forward network with a unique sigmoid slope value (mean of 20 runs). The validation accuracy peaks at a slope of 0.23. Both the Synthetic 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 (in total 15), 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 for test points on the left of the training data, the NN model is unable to infer that a peak exists with a dramatic falloff to the left of it. 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 with a falloff to the left and its approximate location.

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 (despite their behavior being drastically different than what would be expected solely based on the points in the blue shaded region). 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. The main result is that Petri dish obtains highly competitive results even in a limited-compute setting.

Interestingly, these results suggest that it is indeed possible to extract a motif from a larger setting and create a con-

trolled 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. Many more variants of this idea remain to be explored beyond its initial conception in this paper.

The next section describes the related work in the area of NAS speed-up techniques.

2. Related Work

NAS methods have discovered novel architectures that significantly outperform hand-designed solutions (Zoph & Le, 2017; Elsken et al., 2018; Real et al., 2017). These methods commonly explore the architecture search space with either evolutionary algorithms (Suganuma et al., 2017; Miiikkulainen et al., 2018; Real et al., 2019; Elsken et al., 2019) or reinforcement learning (Baker et al., 2016; Zoph & Le, 2017). Because running NAS with full ground-truth evaluations can be extremely expensive (i.e. requiring many thousands of GPU hours), more efficient methods have been proposed. For example, instead of evaluating new architectures with full-scale training, heuristic evaluation can leverage training with reduced data (e.g. sub-sampled from the domain of interest) or for fewer epochs (Baker et al., 2017; Klein et al., 2017).

More recent NAS methods such as DARTS (Liu et al., 2018b) and ENAS (Pham et al., 2018) exploit sharing weights across architectures during training to circumvent full ground-truth evaluations. However, a significant drawback of such weight sharing approaches is that they constrain the architecture search space and therefore limit the discovery of novel architectures.

Another approach to accelerate NAS is to train a NN-based performance prediction model that estimates architecture performance based on its structure (Liu et al., 2018a). Building on this idea, Neural Architecture Optimization (NAO) trains a LSTM model to simultaneously predict architecture performance as well as to learn an embedding of architectures. Search is then performed by taking gradient ascent steps in the embedding space to generate better architectures. NAO is used as a baseline for comparison in Experiment 4.2.

Bayesian optimization (BO) based NAS methods have also shown promising results (Kandasamy et al., 2018; Cao et al., 2019). BO models the architecture space using a Gaussian process (GP), although its behavior is sensitive to the choice of a kernel function that models the similarity between any two architectures. The kernel function can be hand-designed (Kandasamy et al., 2018) or it can be learned (Cao et al., 2019) by mapping the structure of the architecture into an embedding space (similar to the mapping in NAO).

Such kernel functions can potentially be learned using the Synthetic Petri Dish as well.

One NAS method that deserves special mention is the similarly-named *Petridish* method of Hu et al. (2019). We only became aware of this method as our own Synthetic Petri Dish work was completing, leaving us with a difficult decision on naming. The methods are different – the *Petridish* of Hu et al. (2019) is a method for incremental growth as opposed to a learned surrogate in the spirit of the present work. However, we realize that giving our method a similar name could lead to unintended confusion or the incorrect perception that one builds on another. At the same time, the fundamental level at which the use of a Petri dish in biology motivated our approach all the way from its conception to its realization means the loss of this guiding metaphor would seriously complicate and undermine the understanding of the method for our readers. Therefore we made the decision to keep the name Synthetic Petri Dish while hopefully offering a clear enough acknowledgment here that there is also a different NAS method with a similar name that deserves its own entirely separate consideration.

Generative teaching networks (GTNs) also learn synthetic data to accelerate NAS (Such et al., 2020). However, learned data in GTNs helps to more quickly train *full-scale networks* to evaluate their potential on *real* validation data. In the Petri dish, synthetic training and validation instead enables a surrogate microcosm training environment for much smaller extracted motif-networks. Additionally, GTNs are not explicitly trained to differentiate between different networks (or network motifs). In contrast, the Synthetic Petri Dish is optimized to find synthetic input data on which the performance of various architectural motifs is different.

The details of the proposed Synthetic Petri Dish are provided in the next section.

3. 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. Figure 2 provides a high-level overview of the method, which this section explains in detail.

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}_{true}^i , where $i \in 1, 2, \dots, N$) 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.

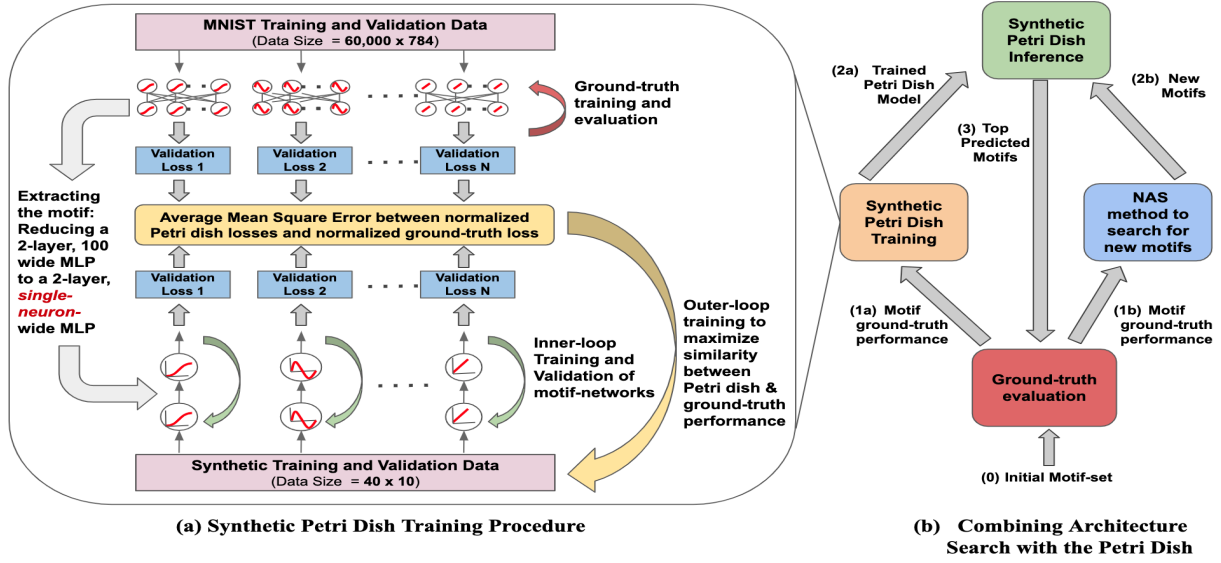


Figure 2. (a) **Synthetic Petri Dish Training.** The left figure illustrates the inner-loop and outer-loop training procedure. The motifs (in this example, activation functions) are extracted from the full network (e.g a 2-layer, 100 wide MLP) and instantiated in separate, much smaller motif-networks (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 mean squared error 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 the Petri Dish.** Functions are depicted inside rectangles and function outputs are depicted as arrows with their descriptions adjacent to them. At step b.0, an initial motif-set is trained and evaluated. The ground-truth performance for this initial set is used for Petri dish model training (step b.1a) and also to generate a set of new motifs through an NAS method like a GA or NAO (step b.1b). The trained Petri dish (step b.2a) model predicts the relative performance of the newly generated motifs (step b.2b) by running Petri dish model inference. A small subset of motifs, with the best predicted performance, are selected for ground-truth evaluation (a repetition of step b.0). These steps are repeated for several iterations and the architecture with the best ground-truth performance is obtained.

3.1. Training the Synthetic Petri Dish

To train the Synthetic 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 performed 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-neuron 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}_{true}^i). 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.1.

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).

$$\theta_{t+1}^i = \theta_t^i - \alpha \nabla \mathcal{L}_{inner_train}(S^{train}, \theta_t^i) \quad i \in 1, 2, \dots, N \quad (1)$$

where α is the inner-loop learning rate and $\theta_0^i = \theta_{init}^i$. 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 ($\mathcal{L}_{inner_valid}^i$). 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. Consider for example what kind of training and validation examples would help to discover convolution. If an identical pattern appears in different training examples in three different quadrants, but not the fourth, the validation data could include an example in the fourth quadrant. Then, any convolutional (or translation invariant) architecture would pass this simple test, whereas others would perform poorly. In this way, generating synthetic training data could potentially have uncovered convolution, the key original architectural motif behind deep learning.

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 setting¹. 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 ($\hat{\mathcal{L}}_{inner_valid}^i$) and the normalized ground-truth validation loss ($\hat{\mathcal{L}}_{true}^i$). The MSE loss is averaged over all the motifs and used to compute a gradient step to improve the synthetic training and validation data.

$$\mathcal{L}_{outer} = \frac{1}{N} \sum_{i=1}^N (\hat{\mathcal{L}}_{inner_valid}^i - \hat{\mathcal{L}}_{true}^i)^2 \quad (2)$$

$$S_{t+1}^{train} = S_t^{train} - \beta \nabla \mathcal{L}_{outer} \quad (3)$$

$$S_{t+1}^{valid} = S_t^{valid} - \beta \nabla \mathcal{L}_{outer} \quad (4)$$

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 of Synthetic Petri Dish training proceeds until the MSE loss converges, resulting in optimized synthetic data.

3.2. Predicting Performance with the Trained Petri Dish

The Synthetic 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

¹We tried an explicit rank-loss as well, but the normalized regression loss performed slightly better empirically.

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; as stated earlier, such relative loss values are sufficient to compare the potential of new candidate motifs. Such Petri dish inference is computationally inexpensive because it involves the training and evaluation of very small motif-networks with very few synthetic examples. Accurately predicting the performance ordering of unseen motifs is contingent on the generalization capabilities of the Synthetic Petri Dish. The ability to generalize in this way is investigated in section 4.1.

3.3. Combining Architecture Search with the Synthetic Petri Dish

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 shows one possible hybridization of Petri dish and NAS, which is the one we experimentally investigate in this work.

First, the Petri dish model is warm-started by training (inner-loop and outer-loop) using the ground-truth evaluation data (P_{eval}) of a small set of randomly-generated motifs (X_{eval}). Then in each iteration of NAS, the NAS method generates M new motifs and the Petri dish inference procedure inexpensively predicts their relative performance. The top K motifs (where $K \ll M$) with highest predicted performance are then selected for ground-truth evaluation. The ground-truth performance of motifs both guides the NAS method as well as provides further data to re-train the Petri dish model. The steps outlined above are repeated for as many iterations as possible or until convergence and then the motif with the best ground-truth performance is selected for the final *test evaluation*.

Synthetic Petri Dish training and inference is orders of magnitude faster than ground-truth evaluations, thus making NAS computationally more efficient and faster to run, which can enable finding higher-performing architectures given a limited compute budget. Further analysis of the compute overhead of the Petri dish model is in section 4.2.

4. Experiments

The first experiment is the easily interpretable proof-of-concept of searching for the optimal slope for the sigmoid, and the second is the challenging NAS problem of finding a high-performing recurrent cell for a language modeling task. The compute used to run these experiments included 20 Nvidia 1080Ti GPUs (for ground-truth training and evalua-

Algorithm 1 Combining Architecture Search with the Synthetic Petri Dish

Input: Set of motifs already evaluated $X = \emptyset$. Initial set of motifs for ground-truth evaluation X_{eval} . Performance of motifs $P = \emptyset$. Number of motifs to generate M . Number of motifs to select K s.t. $K \ll M$. Number of NAS search iterations L . Additional Petri dish hyperparameters (Appendix A.2 and A.3).

for $l = 1$ **to** L **do**

Ground-truth Evaluation: Train each motif in X_{eval} in the full-scale setting and obtain full-scale validation set performance P_{eval} . Enlarge $P : P = P \cup P_{eval}$. Enlarge $X : X = X \cup X_{eval}$.

Petri dish Training: Train the Petri dish model using X and P (see sub-section 3.1).

Architecture Search: Generate a set of M new motifs (X_{new}) using X and P as inputs to the NAS method.

Performance Prediction: Run Synthetic Petri Dish inference for X_{new} (sub-section 3.2) to obtain motif performance predictions.

Selection: Pick top K motifs (denoted by set X_{top}) out of the set X_{new} based on their predicted performance. Set $X_{eval} = X_{top}$.

end for

Output: The motif within X with the best ground-truth performance.

tion) and a MacBook CPU (for Synthetic Petri Dish training and inference).

4.1. 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. The points on the blue curve in Figure 1 demonstrate this fact, where the empirical peak performance is a slope of 0.23. This empirical dependence of performance on sigmoid slope provides an easy-to-understand context (where the optimal answer is known apriori) to clearly illustrate the advantage of the Synthetic Petri Dish. In more detail, we now explain an experiment in which sigmoid slope is the architectural motif to be explored in the Petri dish.

A subset of 30 ground-truth points (blue points in Figure 1, such that each ground-truth point is a tuple of sigmoid slope and the validation accuracy of the corresponding ground-truth MNIST network) are randomly selected from a restricted interval of sigmoid slope values (the blue-shaded region in Figure 1). These 30 ground-truth points are split into two equal parts to create training (15) and validation (15) datasets for both the Synthetic Petri Dish and the NN-

based prediction model. The remaining ground-truth points (outside the blue-shaded region) are used only for testing.

We compare Synthetic Petri Dish to the control of training a neural network surrogate model to predict performance as a function of sigmoid slope. This NN-based surrogate control is a 2-layer, 10-neuron-wide feedforward network that takes the sigmoid value as input and predicts the corresponding MNIST network validation accuracy as its output. A mean-squared error loss is computed between the predicted accuracy and the ground-truth validation accuracy, and the network is trained with the Adam optimizer. Hyperparameters for the NN-based model such as the network’s size, training batch-size (15), epochs of training (50), learning rate (0.001) and scale of initial random weights (0.1) were selected using the validation points. The predictions from the NN-model are normalized and plotted as the red-curve in Figure 1. Results demonstrate that the NN-based model overfits the training points, and poorly generalizes to the test points, completely failing to model the falloff to the left of the peak past the left of the blue-shaded region. As such, it predicts that performance is highest with a sigmoid slope near zero, whereas in fact performance at that point is extremely poor.

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}). Thus, there are 15 motif-networks, one for each training point (the setup is similar to the one shown in Figure 2a). The input and output layers of the motif-network have a width of 10 each. A batch of synthetic training data (size 20×10) and a batch of synthetic validation data (size 20×10) are initialized to the same initial random value (i.e. $S_{train} = S_{valid}$). The motif-networks are trained in the inner-loop for 200 SGD steps and subsequently their performance on the synthetic validation data ($\mathcal{L}_{inner_valid}^i$) is used to compute outer-loop MSE loss w.r.t the ground-truth performance (as described in section 3.1). A total of 20 outer-loop training steps are performed. The model hyperparameters such as number of inner and outer-loop steps, size of the motif-network, size of synthetic data, inner and outer-loop learning rates (0.01 and 0.5 respectively), initial scale of motif-network random weights (1.0) are selected so that the MSE loss for 15 validation points is minimized (hyperparameter selection and other details for the two models are further described in Appendix A.2).

During validation (and also during testing) of Petri dish, new motifs are instantiated in their own motif-networks and trained for 200 inner-loop steps. Thus, unlike the NN-based model that predicts the performance of new motifs based on their scalar value, the Synthetic Petri Dish trains and evaluates each new motif independently with synthetic data (i.e.

it actually *uses* a NN with this particular sigmoidal slope in a small experiment and thus should have better information regarding how well this slope performs). The normalized predictions from the Synthetic Petri Dish are plotted as the green curve in Figure 1. The results show that Synthetic Petri Dish predictions accurately infer that there is a peak (including the falloff to its left) and also its approximate location. The NN surrogate model that is not exposed to the region with the peak could not be expected to infer the peak’s existence because its training data provides no basis for such a prediction, but the Synthetic Petri Dish is able to predict it because the motif itself is part of the Synthetic Petri Dish model, giving it a fundamental advantage, which is starkly illustrated in Figure 1.

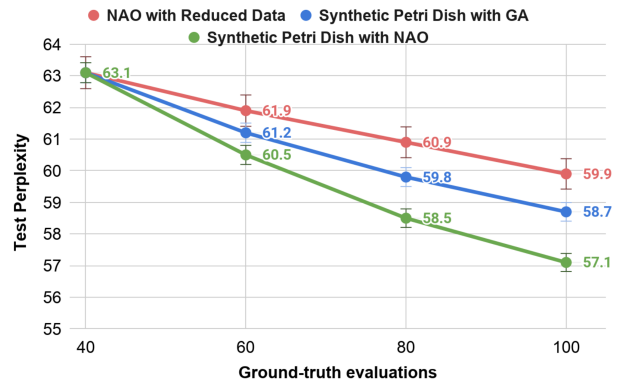


Figure 3. RNN Cell Search for PTB: This graph plots the test perplexity (mean of five runs) of the best found cell for three NAS methods across variable numbers of NAS iterations. All three methods are warmed up at the beginning (Step 0 in Figure 2b) with 40 ground-truth evaluations – notice the top-left point with best test perplexity of 63.1. For **Synthetic Petri Dish with GA** (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 (Luo et al., 2018) (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 fewer 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 after 100 ground-truth evaluations.

4.2. 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 (Marcus et al., 1993). 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 (Luo et al., 2018). This NAS problem is challenging because the search space is expansive and few solutions perform well (Pham et al., 2018). Each cell in the search space is composed of 12 layers (each with the same 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 the PTB data-set for 600 epochs and takes 10 GPU hours on an 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*). Given the cell structure as input, NAO trains an LSTM model to encode and decode the cell structure, and also to predict its performance. Once trained, gradient ascent steps can be taken in the cell embedding space to optimize for RNN cells with higher performance. Thus, NAO consists of a performance-prediction model along with an implicit architecture optimization mechanism. In the published results (Luo et al., 2018), NAO is warm-started by training it on ground-truth evaluation data from 600 random RNN cells. In successive NAS iterations, NAO generates 200 new architectures, *all of which* are then evaluated for their ground-truth performance. At the end of three NAS iterations in which 1,000 ground-truth evaluations are performed (requiring 300 GPU days), NAO discovers 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. Note that NAO can also be combined with weight-sharing (Pham et al., 2018) to improve its efficiency; however, to isolate the effect of the methods themselves from added factors such as weight-sharing (which can also be added in future work to the Synthetic Petri Dish), only methods without weight-sharing are compared.

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 un-

changed). 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 4.1, and it adds negligible compute cost (2 additional hours for training, and a few minutes for inference on a MacBook CPU).

Following the steps outlined in algorithm 1 and Figure 2b, the Petri dish surrogate can be combined with two existing NAS methods: (1) genetic algorithms (GAs) or (2) NAO itself, resulting in two new methods called Synthetic Petri Dish-GA and Synthetic Petri Dish-NAO.

GA-based NAS methods have recently been found successful in several problem domains (Real et al., 2019; Elsken et al., 2019). In Synthetic Petri Dish-GA, RNN cell connectivity is represented as a string of numbers and therefore can be easily modified with GA operators such as point-wise mutation (rate= 0.05) and crossover (rate= 0.3) to generate new cells/motifs. Unlike GAs, the NAO method performs gradient-based search for new motifs. The gradients are potentially noisy and can result in low-performing motifs. Therefore, in principle adding the Synthetic Petri Dish to NAO can help filter of such low-performing motifs (thereby avoiding wasteful ground-truth evaluations). The details of hyperparameter selection for Synthetic Petri Dish-GA and Synthetic Petri Dish-NAO are provided in Appendix A.3.

For the Synthetic Petri Dish variants, at the beginning of search, both the Petri dish surrogate and the NAS method (GA/NAO) used within the Petri dish variant are warm-started with the ground-truth data of an initial motif set (size 40). In each NAS iteration, 100 newly generated motifs (variable M in algorithm 1) are evaluated using the Petri dish inference procedure and only the top 20 predicted motifs (variable K in algorithm 1) are evaluated for their ground-truth performance. 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-GA 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-GA 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). The structure of this discovered cell is shown in Appendix A.3.

5. Discussion and Conclusions

In the general practice of science, often the question arises of what factor accounts for an observed phenomenon. In the real world, with all its intricacy and complexity, it can be difficult to test or even formulate a clear hypothesis on the relevant factor involved. For that reason, often a hypothesis is formulated and tested in a simplified environment where the relevant factor can be isolated from the confounding complexity of the world around it. Then, in that simplified setting it becomes possible to run rapid and exhaustive tests, as long as there is an expectation that their outcome might correlate to the real world. In this way, the Synthetic Petri Dish is a kind of microcosm of a facet of the scientific method, and its synthetic data is the treatment whose optimization tethers the dynamics within the simplified setting to their relevant counterparts in the real world.

By approaching architecture search in this way as a kind of question-answering problem on how certain motifs or factors impact final results, we gain the intriguing advantage that the prediction model is no longer a black box. Instead, it actually contains within it a critical piece of the larger world that it seeks to predict. This piece, a motif cut from the ground-truth network (and its corresponding learning dynamics), carries with it from the start a set of priors that no naive black box learned model could carry on its own. These priors pop out dramatically in the simple sigmoid slope experiment – the notion that there is an optimal slope for training and roughly where it lies emerges automatically from the fact that the *sigmoid slope itself* is part of the Synthetic Petri Dish prediction model. In the later NAS for recurrent cells, the benefit in a more complex domain also becomes apparent, where the advantage of the intrinsic prior enables the Synthetic Petri Dish to have better performance than a leading NAS method when holding the number of ground-truth evaluations constant, and achieves roughly the same performance with 1/10th the compute when allowing differing numbers of ground-truth evaluations.

These benefits ultimately concern both computation and accuracy. Because the Synthetic Petri Dish contains a tiny version of the full problem, candidate architectures can be tested rapidly. Yet we do not give up the value of the surrogate in exchange for such efficiency; on the contrary, because the tiny model contains a piece of the real network (and hence enables testing various hypothesis as to its capabilities), the predictions are built on highly relevant priors that lend more accuracy to their results than blank-slate black box models.

It is also possible that other methods can be built in the

future on the idea of extracting a component of a candidate architecture and testing it in another setting. The opportunity to tease out the 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 that could capture the spirit and power of the scientific process itself.

References

- Adhya, D., Annuario, E., Lancaster, M. A., Price, J., Baron-Cohen, S., and Srivastava, D. P. 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>.
- Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. 2016. URL <https://arxiv.org/pdf/1611.02167v2.pdf>.
- Baker, B., Gupta, O., Raskar, R., and Naik, N. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.
- Cao, S., Wang, X., and Kitani, K. M. Learnable embedding space for efficient neural architecture compression. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=S1xLN3C9YX>.
- Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *CoRR*, abs/1808.05377, 2018.
- Elsken, T., Metzen, J. H., and Hutter, F. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019.
- Hu, H., Langford, J., Caruana, R., Mukherjee, S., Horvitz, E., and Dey, D. Efficient forward architecture search. *Neural Information Processing Systems*, 2019.
- Kandasamy, K., Neiswanger, W., Schneider, J., Póczos, B., and Xing, E. P. Neural architecture search with bayesian optimisation and optimal transport. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 2016–2025. Curran Associates, Inc., 2018.
- Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. Learning curve prediction with bayesian neural networks. In *ICLR*, 2017.

- Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.
- Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018a.
- Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.
- Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in neural information processing systems*, pp. 7816–7827, 2018.
- Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2), 1993.
- Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. Evolving deep neural networks. In Kozma, R., Alippi, C., Choe, Y., and Morabito, F. C. (eds.), *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam: Elsevier, 2018.
- Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.
- Ramachandran, P., Zoph, B., and Le, Q. Searching for activation functions. 2018. URL <https://arxiv.org/pdf/1710.05941.pdf>.
- Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. In *ICML*, 2017.
- Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019.
- Such, F. P., Rawal, A., Lehman, J., Stanley, K., and Clune, J. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data, 2020. URL https://openreview.net/forum?id=HJg_ECEKDr.
- Suganuma, M., Shirakawa, S., and Nagao, T. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17*, pp. 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>.
- Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. 2017. URL <https://arxiv.org/abs/1611.01578>.

A. Appendix

A.1. Petri dish Hyperparameter Selection Procedure

Recall that an initial sampling of candidate motifs evaluated in the ground-truth setting serve as training data for Petri dish (Step 0 in Figure 2b). In order to determine the ideal hyperparameters for Petri dish training, this initial motif ground-truth set (referred to as \mathcal{L}_{true}^i in Section 3) is split into equal parts as training ($\mathcal{L}_{true_train}^i$) and validation ($\mathcal{L}_{true_valid}^i$) sets. Hyperparameter setting that results in the smallest outer-loop MSE loss (\mathcal{L}_{outer}^i in equation 2) on the validation set ($\mathcal{L}_{true_valid}^i$) after training the Petri dish on the training set ($\mathcal{L}_{true_train}^i$), are selected for the full experiment. Specific hyperparameter values for each experiment are provided in the following sections.

A.2. Experimental Setup for Optimal Sigmoid Slope Search

This section provides further details on experiment 4.1. The sigmoid slope value is the motif in this experiment.

A.2.1. SETUP FOR GROUND-TRUTH TRAINING

Previous work has shown that the slope of the sigmoid activation is a critical factor in determining network performance (Ramachandran et al., 2018). Similarly, in this paper, it was empirically found that the validation performance of a 2-layer 100-wide feedforward network on MNIST dataset is dependent on the sigmoid slope. This dependence is shown by the blue-points in Figure 1. To generate each of the blue points, the feedforward network was trained on 50K MNIST training samples and evaluated on 10K validation samples, with the sigmoid slope value ranging between 0.01 – 2.01. All other hyperparameters were held constant during each run and their specific values are provided in Table 1. For each slope value, a mean performance from 20 such runs (along with standard error bars) is plotted in Figure 1.

Hyperparameter	Search Range	Final Setting
Optimizer	N/A	Adam
Inner-loop learning rate	0.001 – 0.01	0.01
L2 weight penalty	1e-5 – 1e-3	1e-5
Number of Epochs	40 – 60	50
Batch Size	20, 40, 50, 100	50

Table 1. Hyperparameter setting for training a 2-layer, 100-wide feedforward network to obtain ground-truth performance in Experiment 4.1.

A.2.2. SETUP FOR PETRI DISH TRAINING AND VALIDATION

The hyperparameter selection procedure follows the same template as described in Section A.1. A subset of 30 ground-truth points are randomly selected from a restricted interval of sigmoid slope values ranging between 0.37 – 1.50 (blue-points in the blue-shaded region of Figure 1). These 30 ground-truth points are split into two equal parts to create training (15) and validation data-set (15) for Petri dish hyperparameter selection. Hyperparameter search range and the final selected values are listed in Table 2.

Hyperparameter	Search Range	Final Setting
Inner-loop Optimizer	N/A	Adam
Motif-network input size	10	10
Motif-network output size	10	10
Motif-network hidden size	1, 3, 5	1
Synthetic training samples	10, 20	10
Inner-loop learning rate	0.001 – 0.01	0.01
Inner-loop L2 penalty	1e-5 – 1e-3	1e-5
Inner-loop training steps	200, 250	250
Outer-loop Optimizer	N/A	Adam
Outer-loop learning rate	0.01 – 0.05	0.05
Outer-loop learning rate decay	0.4 – 0.8	0.4
Outer-loop L2 penalty	1e-5 – 1e-3	1e-5
Outer-loop training steps	20, 40, 60	60

Table 2. Hyperparameter setting for Petri dish training and inference in Experiment 4.1. Because the number of synthetic training samples is small (20), they are used in a single batch for Petri dish inner-loop training. The number of synthetic training samples is the same as the number of synthetic validation samples.

At the beginning of Petri dish training, both the motif-networks weights and the synthetic training/validation data are initialized to random values (drawn from normal distribution). The relative performance of motif-networks after inner-loop training on such random synthetic data is shown in Figure 4. This plot highlights that motif-network training extracts useful prior information about the corresponding motifs.

Implementation Details: As described in Section 3, during Petri dish training, the motif-networks are independently trained and evaluated. Such independent training and evaluation is usually achieved by distributing network training on GPUs. However, because the motif-networks are very small (with only 22 parameters in each) and they share the same synthetic data, their training can be parallelized by a simple trick. The individual motif-networks can be combined to create a single super-network. With appropriate masking connections that prevent any forward and backward propagation across motif-networks within a super-network, we can

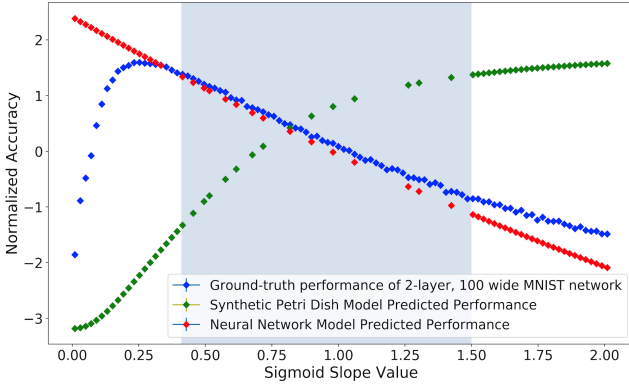


Figure 4. Relative performance of motif-networks with random synthetic data. The green curve shows the performance of motif-networks after inner-loop training on random data. This plot is similar to Figure 1 except that there is no Petri dish outer-loop training in this case.

ensure that all the motif-networks are independently trained at the same time during the super-network training. This parallelization trick allows us to carry out Petri dish training and inference on a basic MacBook CPU very quickly.

A.2.3. SETUP FOR NN-BASED MODEL TRAINING

The training and validation data for the NN-based model is exactly the same as the one used for Petri dish i.e. a set of 30 ground-truth points, where each point is a tuple of sigmoid slope and the validation accuracy of the corresponding ground-truth MNIST network. Similarly to the Petri dish, the NN-based model is trained to predict the *normalized* ground-truth performance (see **Outer-loop training** in Section 3). However, unlike Petri dish that trains and evaluates the motif (i.e. sigmoid slope) in a synthetic setting, the NN-based model takes the real-valued slope directly as its input. A mean squared error is computed between the NN predicted output and the *normalized* ground-truth performance. Hyperparameter search range and their final selected values are listed in Table 3.

A.3. Experiment Setup for Recurrent Cell Architecture Search

This section provides further details on experiment 4.2. The recurrent cell design is the motif in this experiment.

A.3.1. SETUP FOR GROUND-TRUTH EVALUATION

Since NAO (Luo et al., 2018) is used as a baseline method for our experiments, the ground-truth training and evaluation procedure outlined in that paper is also followed here. The final test evaluation of the best found cell (output of Algorithm 1) is also carried out using the setting described

Hyperparameter	Search Range	Final Setting
Optimizer	N/A	Adam
Network hidden size	5, 10, 15, 20	10
Initial Learning rate	0.001 – 0.01	0.01
Learning rate decay	0.8 – 0.99	0.97
Learning rate decay steps	50 – 120	100
L2 weight penalty	1e-5 – 1e-3	1e-4
Number of training steps	100, 150, 200	150
Batch Size	15	15

Table 3. Hyperparameter setting for training the NN-based performance prediction model in Experiment 4.1.

in the NAO paper.

A.3.2. SETUP FOR PETRI DISH TRAINING AND VALIDATION

The hyperparameter selection procedure follows the same template as described in Section A.1. A set of 40 randomly selected cells are evaluated for their ground-truth performance. These 40 ground-truth points are split into two equal parts to create training (20) and validation data-set (20) for Petri dish hyperparameter selection. Hyperparameter search range and their final selected values are listed in Table 4.

Hyperparameter	Search Range	Final Setting
Inner-loop Optimizer	N/A	Adam
Motif-network input size	10	10
Motif-network output size	10	10
Motif-network hidden size	1, 3, 5	3
Synthetic training size	10, 20	20
Inner-loop learning rate	0.001 – 0.01	0.01
Inner-loop L2 penalty	1e-5 – 1e-3	1e-5
Inner-loop training steps	50, 100	50
Outer-loop Optimizer	N/A	Adam
Outer-loop learning rate	0.01–2.5	2.0
Outer-loop learning rate decay	0.4 – 0.8	0.5
Outer-loop L2 penalty	1e-6 – 1e-3	5e-5
Outer-loop training steps	100, 200, 300	200

Table 4. Hyperparameter setting for Petri dish training and inference in Experiment 4.2.

Implementation Details: The parallelization trick of combining multiple motif-networks into a single super-network as described in Appendix A.2.2, is also utilized here as well. Experiment 4.1 required only a single NAS iteration i.e. training Petri dish with a fixed number of training

motifs (and their corresponding ground-truth values) and then predicting the performance of test motifs. Unlike Experiment 4.1, in Experiment 4.2, the number of training motifs increase with additional ground-truth evaluations in each NAS iteration. For example, at the end of three such NAS iterations, there are 100 training motifs that can be utilized for Petri dish training. A growing number of motif-networks during Petri dish training could require further hyperparameter tuning. To avoid such fine-tuning, the total number of motif-network instances during Petri dish training is kept fixed at 40 (thereby also limiting the size of the super-network). During each outer-loop training step, a new batch of 40 motif-networks are randomly sampled from the full-set of available motif-networks.

A.3.3. SETUP FOR NAO AND GA

In Experiment 4.2, two variants of NAO are evaluated in limited resource setting – NAO with Reduced data (red-curve in Figure 3) and Synthetic Petri Dish-NAO (green curve in Figure 3). For both the variants, the NAO model is trained with the same code and hyperparameter setting as in the published work (Luo et al., 2018).

An elitist genetic algorithm (GA) is used to generate new motifs for the Synthetic Petri Dish-GA method (blue curve in Figure 3). In each NAS iteration, the top 20 motifs (with the best ground-truth values) are mutated and recombined to generate 100 new motifs. Each motif is represented as a fixed-size string of numbers that encode the cell connectivity and the types of non-linearity within the cell. During the mutation of a motif, every location within its string is modified with a probability of 0.05. During recombination, two randomly selected motifs are crossed-over at a random location in their strings with a probability of 0.3.

The best found cell using the Synthetic Petri Dish-NAO method has a test perplexity of 57.1 and is shown in Figure 5.

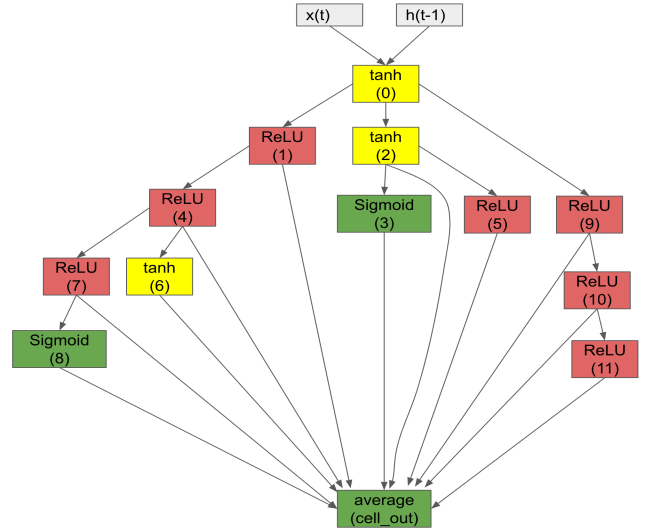


Figure 5. Best Cell found by Synthetic Petri Dish-NAO method.