Supplemental material for

Neural networks embrace learned diversity

Anshul Choudhary,[1] Anil Radhakrishnan,[1] John F. Lindner*,[1,2] Sudeshna Sinha,[3] and William L. Ditto[1]

[1] *Nonlinear Artificial Intelligence Laboratory, Physics Department, North Carolina State University, Raleigh, NC 27607, USA*
[2] *Physics Department, The College of Wooster, Wooster, OH 44691, USA*
[3] *Indian Institute of Science Education and Research Mohali, Knowledge City, SAS Nagar, Sector 81, Manauli PO 140 306, Punjab, India*

(Dated: April 12, 2022)

CONTENTS

I. MULTIPLE LAYERS

We have explored learned diversity using both multiple hidden layers in the classifying neural network and multiple hidden layers in the neuronal sub-networks with similar results, as summarized by Fig. 1, which plots mean validation accuracy versus training number in epochs after meta-learning two activation functions based on a sinusoid using networks of 1, 2, 3 hidden layers. In this example, the heterogeneous network (thick line) outperforms *both* homogeneous networks of either component (thin solid lines) *and* homogeneous networks using popular activation functions (dashed lines) on average.

Hyperparameters are the same in each case, with no attempt to optimize for additional layers. Accuracies are modest due to the small network sizes and the inherent difficulty of classifying MNIST-1D digits, which is challenging even for humans, but the modest accuracies allow us to clearly illustrate the learned diversity improvements.

For 1 hidden layer, the meta-learning inner loop trained a $40 : 100 : 10$ fully-connected feed-forward neural network with 10^4 40-pixel classified digit images shuffled 5 times while the meta-learning outer loop updated the weights and biases of the 2-activation-function $1 : 50 : 1$ sub-networks 10^3 times, resetting the inner network every time, and similarly for multiple hidden layers.

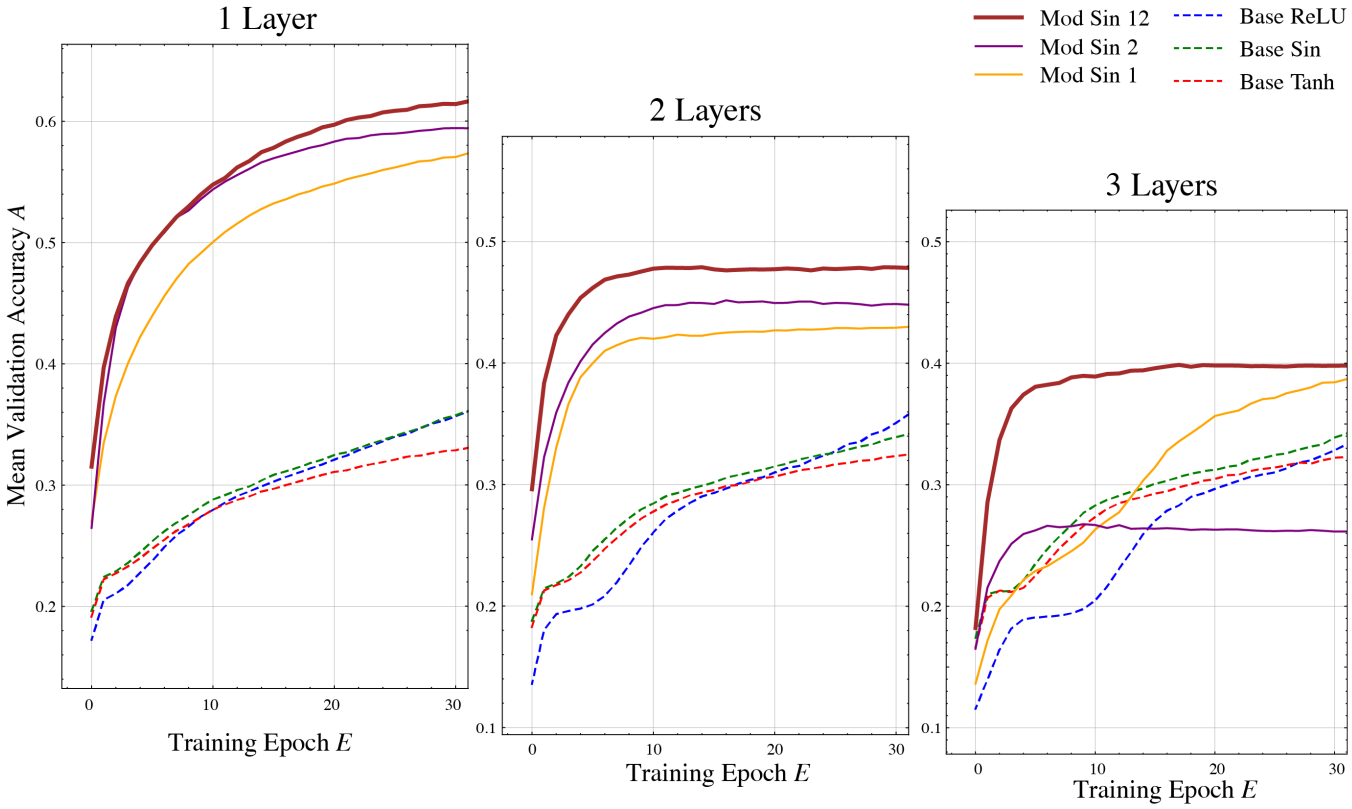

FIG. 1. Multiple hidden layers. Example mean validation accuracy versus number of training data in epochs after meta-learning two sinusoid-based activation functions using networks of 1, 2, 3 hidden layers. Heterogeneous network (thick line) outperforms homogeneous networks of either component (thin solid lines) and homogeneous networks using popular activation functions (dashed lines) on average.

II. NEURON NUMBER DETAILS

As reported in the main manuscript, we have explored learned diversity by systematically varying the number of hidden-layer neurons in our neural networks, as summarized by the Fig. 2 example. We used the same learning rate for each network size, optimized to avoid over-fitting. We manually fitted the learned activation functions, evolved from zero, for each of the configurations. We then ran them for 15 epochs for 100 different realizations and computed their classification accuracies. For all sizes, the mixed networks outperform the pure networks on average and also outperform both a learned single activation function and traditional activation functions like ReLU and sine.

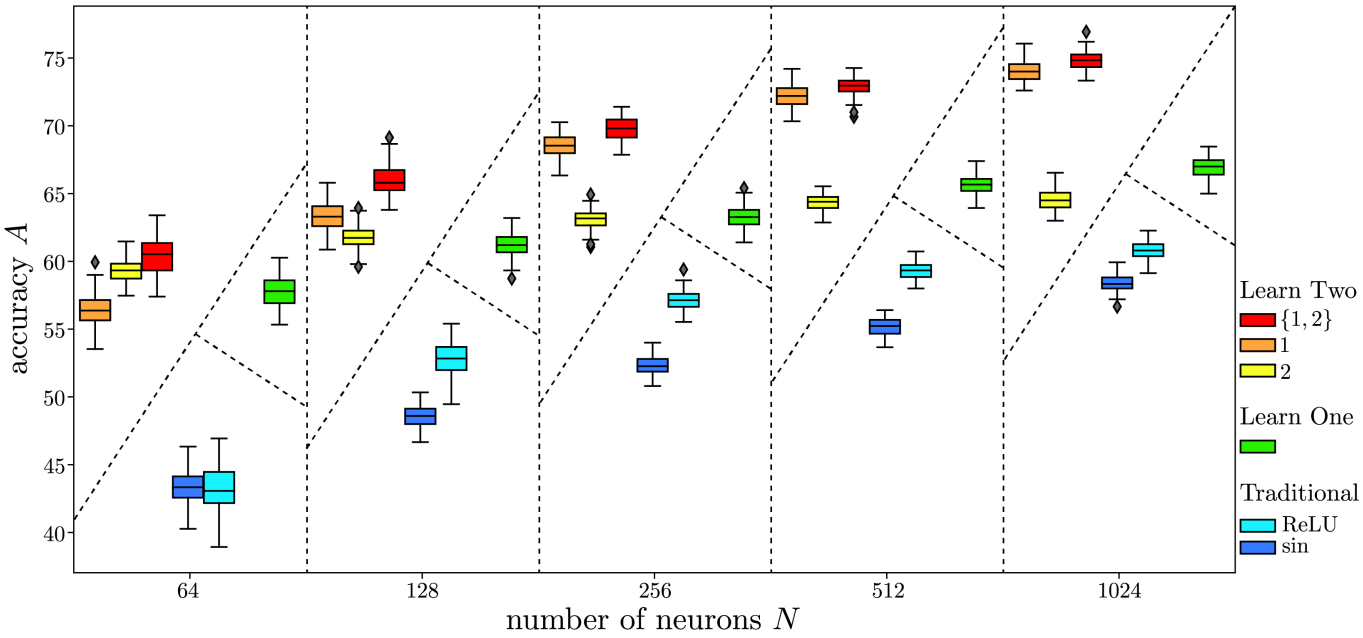

FIG. 2. Neural network MNIST-1D classification accuracy as a function of network size. Box and whiskers plots summarize accuracy distribution (including median, quartiles, extent, and outliers). Learning rate is optimized to avoid over-fitting but is the same for all network sizes. Activation functions evolved from zero (the null function) with similar results evolved from sine. Mixed networks of 2 neuron types outperform pure networks on average for all sizes and outperforms both single learned activation and traditional activations.

III. META-LEARNING 3 ACTIVATIONS

Figure 3 summarizes meta-learning the activation functions of neurons subject to the constraint of having three functions distributed equally among the neuronal population for the MNIST-1D classification task. The meta-learning generates roughly three classes of activations from its sinusoidal (or other) start, typically symmetric or anti-symmetric near the origin, providing a kind of spanning basis, as orthogonal as possible. Again the mixed network outperforms the three pure networks on average.

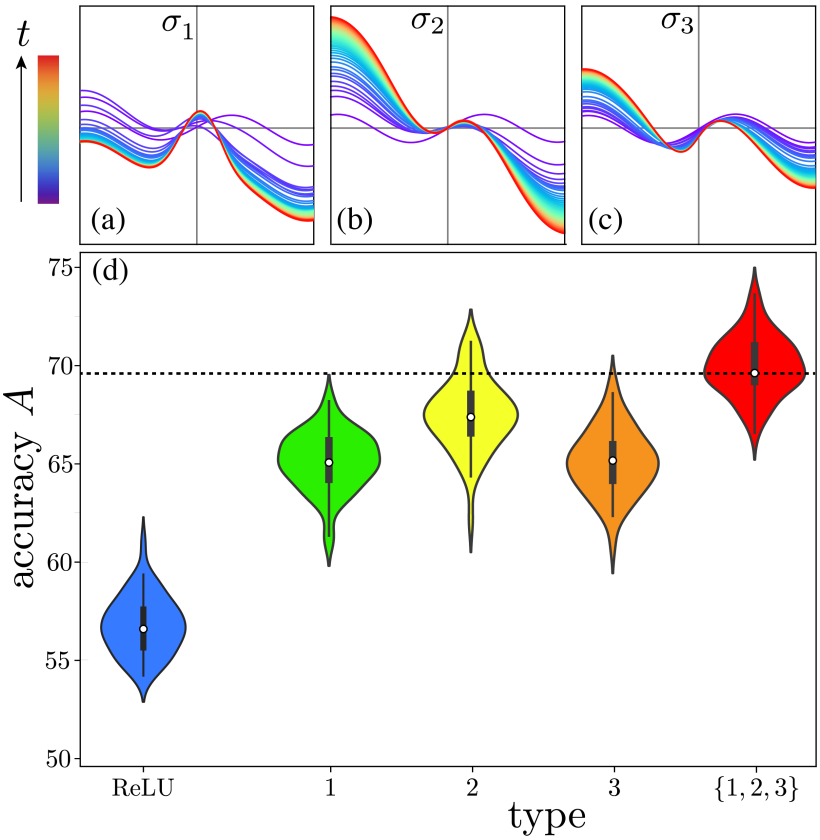

FIG. 3. Meta-learning 3 activations for classification. (a-c) Activation functions $\sigma_n(a)$ evolve from a base sinusoid, with violet-to-red rainbow colors encoding time t. (d) Violin plots summarize validation accuracy A for 50 fully connected neural networks of type-1 neurons (green), type-2 neurons (yellow), type-3 neurons (orange), and a mix of all 3 neuron types (red). The mix of 3 neuron types outperforms any single neuron type on average.

IV. HESSIAN DETAILS

To understand the impact of learned diversity on the geometric nature of the loss function minima, we computed the spectrum of the Hessian matrix $\mathbf{H} = \nabla^2 \mathcal{L}$, which measures the loss function curvature, as summarized by the Fig. 4 spectral plots for meta-learning two activation functions for MNIST-1D classification. Fraction of bounded area in the (red) shaded region is a measure of the flatness of a minimum, as is the hessian trace, both of which trend similarly.

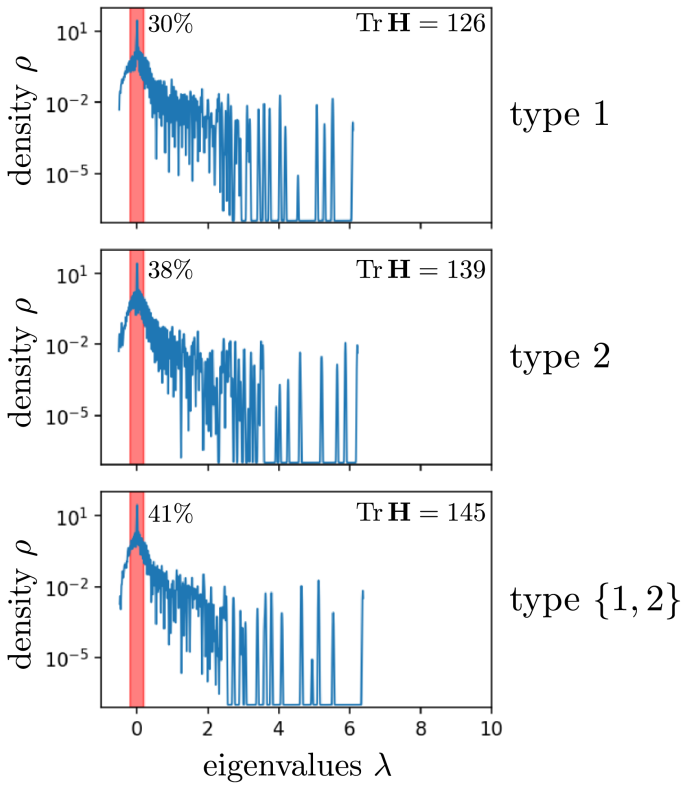

FIG. 4. Spectral density ρ versus eigenvalues λ of the loss-function second-derivative hessian matrices for classifying MNIST-1D with pure and mixed networks display clear trends in both traces and area bounded near zero.

V. UNIVERSALITY OF LEARNED ACTIVATIONS

Learned activation functions appear qualitatively independent of the base activation function, as summarized by Fig. 5, where learned activation functions $\sigma(x)$ based on rectifying ELU(x), harmonic sin(x), and saturating tanh(x) have qualitatively similar near-zero behavior. In (a) a single function is learned that behaves as an odd function near zero. When allowed to learn two activations, as in (b) and (c), odd and even functions are learned for many base functions.

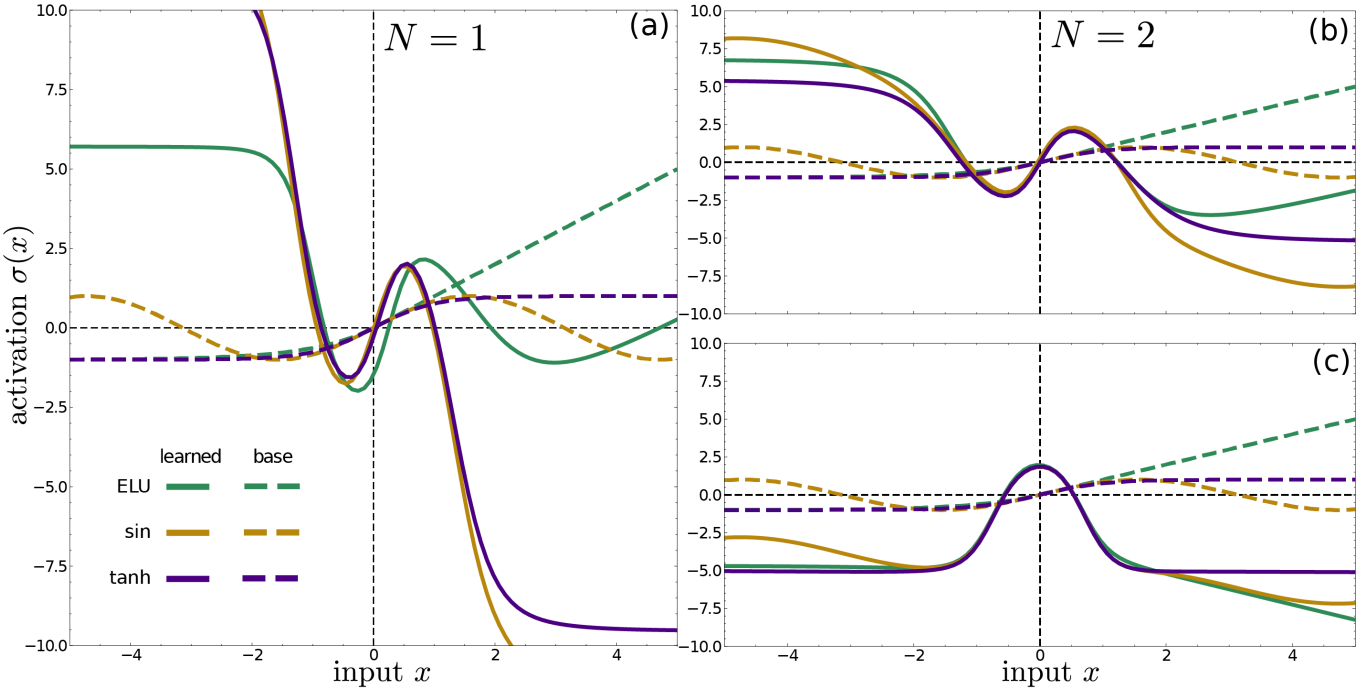

FIG. 5. Learned activation functions for ELU (rectifying), sine (harmonic), tanh (saturating) have qualitatively similar near zero behavior. In (a) a single function is learned that behaves as an odd function near zero. When allowed to learn two activations, as in (b) and (c), odd and even functions are learned starting from many different base functions.

VI. STOCHASTIC PROCESSES INSIGHTS

Optimizing a neural network by randomly shuffling training data is like a noisy descent to a minimum in a potential landscape, as in Fig. 6. The landscape is the network's error or loss as a function of its weights and biases, and its shape depends on the neuron activation functions. The effective dynamics is that of an overdamped particle buffeted by noise sliding on a complicated potential with many local minima. The Langevin equation

$$d\theta_t = -\nabla\mathcal{L}(\theta_t)\,dt + \sqrt{2\mathbf{D}}\cdot d\mathcal{W}_t \tag{1}$$

with noise intensity $\mathbf{D} = \eta\,\mathcal{L}(\theta)\mathbf{H}(\theta^*)/B$ describes the evolution of the weights and biases $\theta = \{W_{ij}, b_i\}$ in a valley with local minimum θ^*, where η is the learning rate and B is the training batch size [1–4]. The drift term dt includes minus the gradient of the loss function $\mathcal{L}$, and the Brownian motion noise term $d\mathcal{W}_t$ includes the learning rate η. The noise aligns with the Hessian near a minimum, and the Eq. 1 Hessian dependence ensures that stochastic gradient descent escapes multiple sharp minima via directions corresponding to large eigenvalues of the Hessian and eventually converges to the flatter minimum.

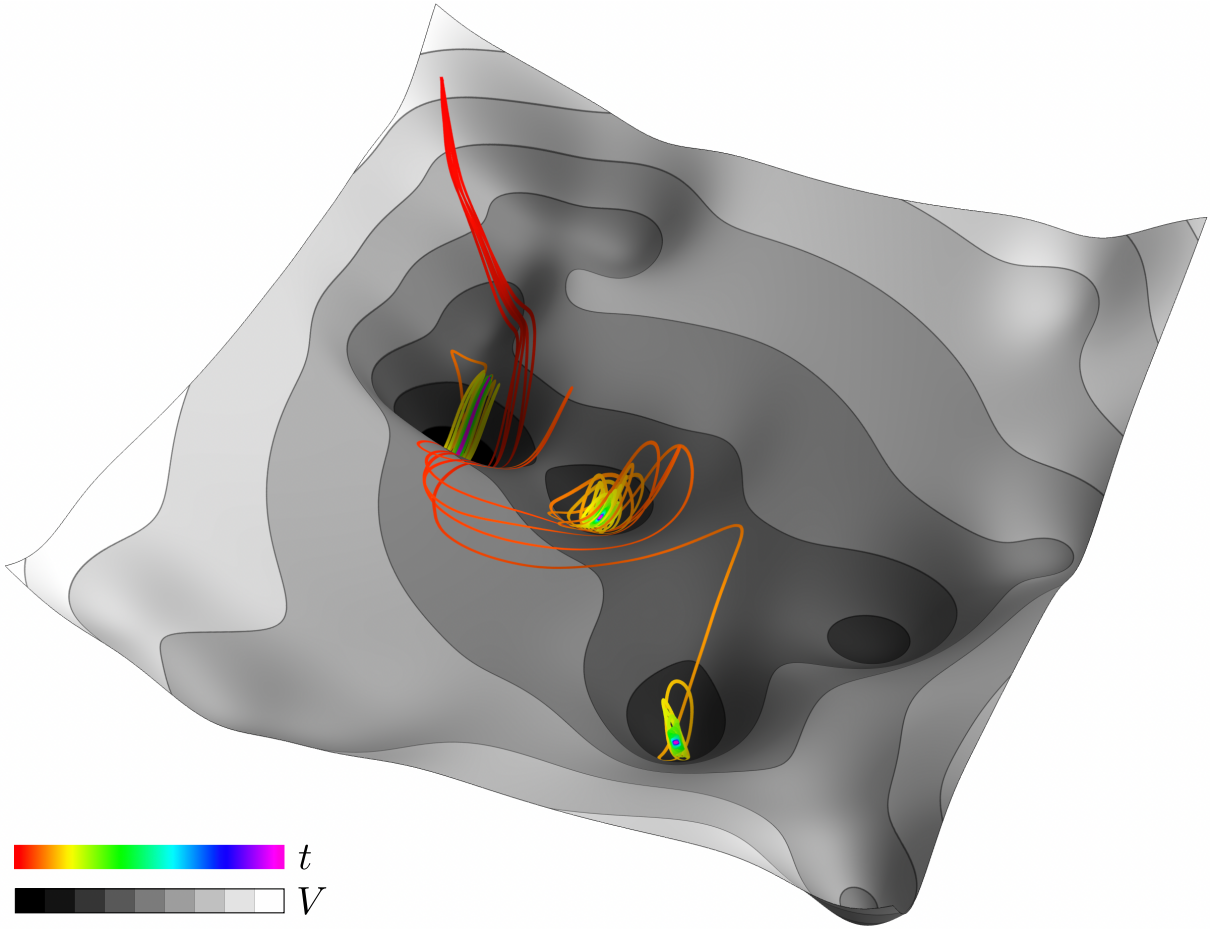

FIG. 6. Noisy descent. Rainbow colors code time t as state point wanders to different local minima of potential landscape V from same initial conditions under multiple realizations of the same noise.

VII. COMPUTER IMPLEMENTATION

A. Overview

We implement our neural networks in the Python programming language using the PyTorch open source machine learning library. The code for the analysis and the network implementation in PyTorch can be found at our GitHub repository [5].

B. Gradient Based Metalearning

The learner-metalearner structure of the metalearning algorithm brings with it significant computational costs. Current implementation of the algorithm is constrained on the number of learner/inner loops within the metalearner/outer iterations since the inner loop is held in memory for the outer loop computation and optimization. In fact, this is one of the fundamental limitations of the gradient based meta-learning algorithm that limits the horizon of meta-objective function [6].

C. Hessian Computation

PyHessian Library is used to compute hessian based statistics without the cost of generating the full hessian matrix. The trace of the hessian matrix is computed using Hutchinson's method exploiting the symmetric nature of the matrix [7]. The Empirical Spectral Density(ESD) of hessian eigenvalues is computed through Stochastic Lanczos Quadrature(SLQ) [8] within several successive approximation schemes. Details can be found in Yao et al. [9]. At an implementation level, a classifier using the learned activation(s) is trained in Pytorch and the model is saved. Using this saved model and test data, PyHessian can use PyTorch's backward graph to compute the gradients needed to build the hessian trace and ESD.

D. Interpolation and Fitting

The activation function is captured after metalearning as the output of the learned activation networks on the interval $[-10, 10]$ with 100 linearly spaced points. This output is then linearly interpolated between points and used as the activation function for the classifer at validation. Quadratic or cubic splines or symbolic regression can also be used. We need high order (> 10) polynomials to fit the activation curves accurately so, while possible, we do not recommend polynomials as a reliable way to capture the features of the learned activation function.

[1] Takashi Mori, Liu Ziyin, Kangqiao Liu, and Masahito Ueda. Logarithmic landscape and power-law escape rate of SGD. *arXiv:2105.09557*, 2021.

[2] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 18:1–35, 2017.

[3] Justin Sirignano and Konstantinos Spiliopoulos. Stochastic gradient descent in continuous time: A central limit theorem. *Stochastic Systems*, 10(2):124–151, 2020.

[4] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

[5] Our code is available at https://github.com/nonlinearartificialintelligencelab/diversityNN.

[6] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

[7] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM*, 58(2):8:1–8:34, April 2011.

[8] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast Estimation of $\mathrm{tr}(f(A))$ via Stochastic Lanczos Quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099, January 2017.

[9] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael Mahoney. PyHessian: Neural Networks Through the Lens of the Hessian. *arXiv:1912.07145*, March 2020.