The BUTTER Zone: An Empirical Study of Training Dynamics in Fully Connected Neural Networks

Member, IEEE, Charles Edison Tripp, Jordan Perr-Sauer, Lucas Hayne, Monte Lunacek, Jamil Gafur

Abstract—We present an empirical dataset surveying the deep learning phenomenon on fully-connected feed-forward multilayer perceptron neural networks. The dataset, which is now freely available online, records the per-epoch training and generalization performance of 483 thousand distinct hyperparameter choices of architectures, tasks, depths, network sizes (number of parameters), learning rates, batch sizes, and regularization penalties. Repeating each experiment an average of 24 times resulted in 11 million total training runs and 40 billion epochs recorded. Accumulating this 1.7 TB dataset utilized 11 thousand CPU core-years, 72.3 GPU-years, and 163 node-years. In surveying the dataset, we observe durable patterns persisting across tasks and topologies. We aim to spark scientific study of machine learning techniques as a catalyst for the theoretical discoveries needed to progress the field beyond energy-intensive and heuristic practices.

Index Terms—Deep Learning, Empirical, Dataset, Hyperparameter, Topology, Architecture, Depth, Width, Shape, Size, Parameter Count, Scientific, Fully-Connected, Multilayer Perceptron, Neural Network, Machine Learning, BUTTER, Performance, Efficiency

I. INTRODUCTION

Historically, the governing dynamics of machine learning methods have been derived from first principles analysis. However, increasingly complex methods, such as deep learning, have proven resistant to such theoretical analysis. Without substantial insight into the governing dynamics of deep learning, practitioners are left with trial-and-error heuristics and energy-intensive hyperparameter optimization schemes. We hypothesize that deep learning methods have attained sufficient complexity to justify their empirical study as a complex computational phenomena, and that scientific study of the behavior of deep learning can provide a guiding light for theoretical analysis. In this spirit, we collected and present an empirical dataset, which we name “BUTTER”, surveying the deep learning phenomenon on fully-connected feed-forward multilayer perceptron neural networks (FC MLPs), encompassing the training and test performance of network topologies sweeping across 13 learning tasks, 8 network shapes, 14 depths, 23 parameter counts, 4 learning rates, 6 batch sizes, 4 label noise levels, 27 regularization penalties, 3 optimizers, and two momentum levels. The dataset tests 483 thousand distinct experiments with an average of 24 repetitions each, yielding 11 million training runs covering 20 performance metrics for each of the 40 billion training epochs observed. Accumulating this 1.7 TB dataset utilized 11 thousand CPU core-years, 72.3 GPU-years, and 163 node-years across four high-performance computing (HPC) systems.

BUTTER studies MLPs as they are a fundamental starting point for deep learning research. As discussed in Section II, an empirical deep learning dataset of BUTTER’s scale is novel, even for MLPs. As findings about MLPs can inform investigations into other other salient network classes including convolutional neural networks (CNNs), Transformers, recurrent neural networks (RNNs), and graph neural networks (GNNs), BUTTER forms a guide-stone for generating and analyzing empirical datasets studying such architectures, which could extend the BUTTER dataset in the future. We hope this analysis contributes to a call-to-arms for researchers to dig deeper and to conduct their own empirical studies to gain a better understanding of this computational phenomenon and to improve deep learning engineering practices.

II. BACKGROUND

We present this dataset to promote investigation into the expressivity, generalization, and trainability of deep learning models, three properties identified by Chen et al. [1] as important for successful deep learning. Early researchers focused primarily on expressivity, proving that FC MLPs serve as universal approximators [2] and bounded the width of networks needed to approximate certain functions [3]. Inspired by practical results suggesting that MLPs can overcome the curse of dimensionality, recent theoretical work identified scaling law bounds for rates of convergence when training on restricted forms of binary classification [4] and regression [5], [6]. The neural tangent kernel (NTK) provides an analytic expression for the first order approximation of training dynamics for single-layer MLPs in the infinite-width case [7]. Other recent theoretical results include proving that functions can be constructed that are expressible by 3-layer MLPs that are not well approximated by 2-layer MLPs with similar sizes [8], Kolmogorov-optimal approximation bounds for MLPs using infinite training data [9], lower bounds on the error of ReLU MLP networks for smooth functions as a function of width and depth [10], a proof that CNNs can outperform one-layer MLPs [11], a proof that rectangular ReLU FC MLPs with a width proportional to their number of weights can achieve an approximation rate for which cannot be attained by shallower networks [12], and bounds on the approximation rates of polynomial and periodic activation function based MLPs [13]. Theories like these outline what is possible in deep learning, but practitioners need reliable predictions of how their design choices will affect their model training and generalization given their practical constraints. A practitioner might ask “If I train this network on my dataset with these hyperparameters, what test loss will it achieve and how many epochs will it take?”

Recent empirical work has begun to address this challenge. Pezeshki et al. [14] analytically reproduce double descent
curves at various regularization strengths and posit the existence of slow and fast features that are learned at different rates. Chen et al. [15] show that for linear regression, the distribution of new features added to the training set determines the location of descents and ascents in the generalization error curve. Adlam et al. [16] present a mathematical foundation for the double descent phenomenon using random matrix theory and predict a complex relationship between the size and generalizability of a model. In 2015 and elaborated in 2017, Neyshabur et al. presented empirical evidence that the classical u-shaped bias-variance trade off [17], [18] is not exactly observed in deep learning [19], [20]. Neal et al. [21] and Belkin et al. [22] discussed empirical and theoretical evidence that the classical bias-variance trade-off may not apply to overparameterized models; beyond the interpolation threshold, we may observe paradoxically decreasing generalization error. In 2019 and 2021 Nakkiran et al. presented empirical evidence of double descent in terms of number of parameters and epochs [23], [24].

In 2020, Kaplan et al. empirically studied scaling laws of neural language models given unbounded training data [25] and Henighan et al. empirically studied scaling laws of Autoregressive Generative Modeling [34]. In 2022, Tay et al. derived scaling laws across inductive biases and model architectures, and summarized many scaling law findings [35]. Although focused on the unbounded data regime, empirical studies of scaling laws suggest performance scales as a power-law with dataset size, training effort, and parameter count. Further, these studies suggest that width, depth, and model shape have only small impacts on performance. Our dataset corroborates these findings for MLPs and finite datasets.

Universal approximation bounds and the double descent hypothesis relate relatively simple measures (layer width and effective model complexity) to a model’s generalization but, in practice, researchers are concerned with many more architectural choices than these two. Jiang et al. [36] conducted an empirical study of complexity measures, or “properties of the trained model, optimizer, and possibly training data” that “monotonically relate to some aspect of generalization.” Across 10,000 trained models, they found promising PAC-Bayes and optimization-based metrics for predicting generalization performance. Dziugaite et al. [37] conducted a similar study seeking distributionally robust generalization measures. Additionally, Novak et al. [38] describe a sensitivity metric based on the norm of the Jacobian of the loss function, and find that robustness in this metric around training data is predictive of generalizability. To help explain which networks can be trained to achieve good generalization scores (trainability), Frankle and Carbin [39] established the lottery ticket hypothesis (LTH) from empirical study. This hypothesis suggests that “SGD seeks out and trains a well-initialized subnetwork” and that “overparameterized networks are easier to train because they have more combinations of subnetworks that are potential winning tickets.”

To enable offline benchmarking of Neural Architecture Search (NAS) algorithms, several related datasets have been generated. Table I compares the BUTTER dataset to seven other NAS benchmarking datasets that provide raw “tabular” results (leaving out benchmarks which only provide a surrogate model and no raw results) [40]. Tabular NAS benchmarks exist for image classification [30], [41], [45], automatic speech recognition [46], and other tasks [47]. In contrast to the goals of NAS research, we present a large empirical dataset to stimulate discoveries of patterns underlying the expressivity, generalization, and trainability of deep learning models. To this end, BUTTER contains more experiments than the NAS datasets, and repeats each experiment using unique random seeds an order of magnitude more times. Repetitions allow us to access information about convergence and performance variability, enabling dataset consumers to study the stochastic nature of deep learning including the LTH. Ignoring the variability of the training process exposes users to the risk of spurious and incorrect interpretations due to sample bias. In some cases, BUTTER experiments minimize test loss far beyond that which is achieved in the 12 to 200 epochs covered by other datasets. The long training runs in BUTTER (one to two orders of magnitude more epochs than other datasets) allow us to observe the scaling laws of over-fitting [33], to escape local minima in the training curve, and to probe for deep double descent [23]. Large detailed sweeps over the configuration space enables dataset consumers to analyze scaling laws as a function of these parameters and to determine how these laws might generalize across tasks. Finally, BUTTER covers 31 regularization levels, a dimension not studied in any of the comparable datasets listed in Table I.

| Dataset            | Experiments | Repetitions | Tasks | Epochs | Width Range | Depth Range | Batch Sizes | Learning Rates | Regularization Levels | Network Class |
|--------------------|-------------|-------------|-------|--------|-------------|-------------|-------------|-----------------|----------------------|----------------|
| BUTTER             | 483k        | 30          | 13    | 39     | 1-7M        | 2-20        | 6           | 4               | 31                   | MLP            |
| NAS-HPO-Bench [25] | 62k         | 4           | 1     | 10     | 16-512      | 2           | 4           | 1               | 1                    | MLP            |
| NAS-HPO-Bench-II   | 192k        | 3           | 3     | 12     | 64          | 15          | 6           | 8               | 1                    | CNN            |
| LCBench [27]       | 2k          | 1           | 35    | 12-50  | 64-1.0k     | 1-5         | n/a         | n/a             | n/a                  | MLP            |
| NATS-Bench         | 6k          | 1           | 3     | 200    | 8-16        | 15          | 1           | 3               | 1                    | CNN            |
| NAS-Bench-101     | 423k        | 3           | 1     | 108    | 128         | 9           | 1           | 1               | 1                    | CNN            |
| NAS-Bench-ASR [31] | 8k          | 3           | 1     | 40     | 600-1.2k    | 12-24       | 2           | 1               | 1                    | LSTM           |
| TransNAS-Bench [32]| 7k          | 1           | 7     | 100    | 64          | 8-12        | 1           | 1               | 1                    | CNN            |

4Experiments in BUTTER run for 3000 epochs, except the 30k and 300 epoch runs as indicated in Table I.
5LCBench randomly sampled batch sizes, learning rates, and regularization levels.
TABLE I

A SUMMARY OF THE SWEEPS COMPOSING THE DATASET. TO CONSERVE SPACE, THE BOUNDS OF EACH RANGE ARE LISTED. SALIENT DIFFERENCES FROM THE PRIMARY SWEEP ARE SHOWN IN BOLD. TASK #’S AND SHAPE #’S ARE INDEXED IN TABLES IV AND III.

| Sweep | Attribute | 300 Epoch | 3k Epoch | Learning Rate | Label Noise | Batch Size | Regularization | Optimizer |
|-------|-----------|-----------|----------|---------------|-------------|------------|----------------|-----------|
| Experiments | 3.88k | 2.88k | 22.9k | 59.5k | 102k | 15.9k | 79.1k | 215k |
| Runs | 1.47M | 2.88k | 239k | 1.32M | 3.69M | 223k | 850k | 4.96M |
| Total Epochs | 4.42B | 4.23M | 7.17B | 3.96B | 11.1B | 669M | 14.9B | 9.81B |
| Repetitions | 30 | 10 | 10 | 20 | 30 | 10 | 20 | 30 |
| Epochs / Run | 3k | 300 | 30k | 3k | 3k | 3k | 3k | 3k |
| Task # | 1.13 | 32 | 1.9 | 1.9 | 1.7 | 1.9 | 1.9 | 1.13 |
| # Parameters | 1.8 | 1.6 | 1.6 | 1.8 | 1.4 | 1.4 | 1 | 1.8 |
| # Parameters | 25 \( ^{-25} \) | 25 \( ^{-25} \) \( ^{-27} \) | 25 \( ^{-25} \) \( ^{-18} \) | 25 \( ^{-24} \) | 25 \( ^{-24} \) | 25 \( ^{-24} \) | 25 \( ^{-25} \) | 25 \( ^{-27} \) |
| Depth | 2.2 | 2 | 2.1 | 2.2 | 2.2 | 2.1 | 2.1 | 2.0 |
| Learning Rate | 10 \( ^{-4} \) | 10 \( ^{-4} \) | 10 \( ^{-4} \) | 10 \( ^{-2} \) \( ^{-5} \) | 10 \( ^{-4} \) \( ^{-5} \) | 10 \( ^{-4} \) | 10 \( ^{-2} \) \( ^{-5} \) | 10 \( ^{-2} \) \( ^{-5} \) |
| Batch Size | 2 \( ^{8} \) | 2 \( ^{8} \) | 2 \( ^{8} \) | 2 \( ^{8} \) | 2 \( ^{8} \) | 2 \( ^{8} \) | 2 \( ^{8} \) | 2 \( ^{8} \) |
| Label Noise | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Regularization | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.02 |
| Momentum | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.02 |
| Optimizer | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

\( ^{a} \) Optimizers are numbered (1) Adam, (2) SGD, and (3) RMSProp.

III. CONTRIBUTIONS

1) We present [BUTTER](https://github.com/NREL/BUTTER-Better-Understanding-of-Training-Topologies-through-Empirical-Results), an empirical deep learning dataset recording 20 performance metrics at each epoch of an average of 24 repetitions of 483 thousand distinct experiments totalling 11 million training runs and 40 billion epochs. Accumulating this 1.7 TB dataset utilized 11 thousand CPU core-years, 72.3 GPU-years, and 163 node-years. We describe this dataset in Section IV and make it freely available under the Creative Commons Attribution-ShareAlike 4.0 International Public License [48].

2) We describe the dataset with tables enumerating the hyperparameter dimensions in Section IV. We visualize and survey the dataset through the use of partial dependence plots of the marginals for each hyperparameter in Section IV-A, making observations and drawing comparisons to findings in related work cited above. We observe certain trends which persist in the data across tasks and topologies, and present these findings in a preliminary analysis in Appendix [A].

3) We make the source code for our empirical machine learning framework [49], distributed job queue [50] which distributes the experiments over as many systems as desired, and the code to reproduce our analysis figures freely available under the MIT Licence. [1]

IV. DESCRIPTION OF THE BUTTER DATASET

The BUTTER dataset consists of seven hyperparameter sweeps over the MLP design space, detailed in Table III. Each sweep is designed to study the impacts of one or more design choices on training and model performance. We repeat each experiment multiple times to capture variations in performance due to the stochastic initialization and optimization. Each repetition of an experiment used a different random seed, and the seed, software versions, and environment used are recorded in the dataset to ensure reproducibility. We refer to each distinct combination of parameters as an “experiment” and each repetition of an experiment as a “repetition”. Due to overlaps between sweeps, the total number of experiments and repetitions in the dataset is less than the sums of these quantities over all sweeps. Network shapes are listed in Table III and define how the width of each layer changes with depth. Learning tasks were drawn from the Penn Machine Learning Benchmarks (PMLB) [51] because PMLB provides a single source MIT Licensed library with a...
Fig. 1. Marginal effects plots for number of parameters, depth, and shape on the Primary Sweep. These marginals plots show the median minimum test loss achieved and the median number of epochs to reach it (y-axes) on a per-task (dashed lines) and overall (thick line) basis as a function of a particular hyperparameter (x-axis). The overall interquartile range is shaded to indicate how much variation in performance might be due to other hyperparameter choices. A thin variation band indicates that the hyperparameter value tightly controls performance and other parameter settings have little impact. A wide band indicates that the setting’s effect is not robust and performance may be largely dictated by other factors. Shape numbers are indexed in Table III.

Fig. 2. Generalization performance depends on the aspect ratio. We trained networks with varying widths and depths to study the effects of these model architecture choices on generalization. Test losses are min-max normalized and log scaled for each task in the Primary Sweep.

We chose the tasks indexed in Table IV to test both classification (7 tasks) and regression (6 tasks) tasks covering a range of observation (sleep’s 106k to 505_tecator’s 240 observations) and feature (banana’s 2 to mnist’s 784 features) counts. By drawing this variety of tasks from PMLB, we aim for patterns identified in our dataset to generalize to a meaningful range of tasks, and to increase the possibility of identifying dynamics specific to particular dataset types. Our dataset does not include any offensive or personally identifiable information. The exact topology used in an experiment is explicitly stored, but is defined by the combination of shape, depth, and number of trainable parameters. Task data was randomly shuffled on every repetition using an 80%/20% training-test split. The Adaptive Moment Estimation optimizer (Adam) was used as the optimizer for all runs outside of the optimizer sweep, and rectified linear unit (ReLU) activation functions were used for all hidden layers in all networks. We used Adam because it has proven effective across many network configurations, choice of learning rate, and tasks due to its step-size adaptation. Adam occupies a middle ground between Momentum, Root Mean Square Propagation (RMSProp), and AdaGrad optimizers. Adam enjoys a long tenure as a popular optimizer and has garnered over 11k citations.

A. Motivations for and observations from each sweep
We began with the “Primary Sweep” to investigate the influence of parameter count, depth, width, shape, and number of epochs on model performance. Given the substantial body of work in NAS, we suspected these factors play critical roles in model performance. This sweep is a grid search spanning 13 learning tasks, 20 different numbers of trainable parameters (network sizes), 8 network shapes, and 14 depths with 30 repetitions of each combination. Each repetition in the Primary Sweep was trained for 3 thousand epochs. Consistent with recent work suggesting that scaling laws of neural networks are roughly

2See https://epistasislab.github.io/pmlb/ for detailed summaries of PMLB datasets, including those used to generate this dataset.

simple API enabling us to study a wide variety of tasks.
power-law relationships, we found that many patterns were better visualized by plotting means and quartiles on semilog- or log-scaled axes rather than on linear scaled axes.

As shown in Figure 1 and Appendix B, Figures 9-12, network shape had a negligible impact on training performance. This finding is consistent with [33], [35], [62], [63] and further study of BUTTER and other datasets could be critical to improving the efficiency of NAS approaches. We also see that parameter count strongly influences the generalization performance of a network, with larger networks performing better, even well beyond the interpolation threshold. And, larger networks maximize performance in fewer epochs, but both effects have diminishing returns (note the log-scaled x-axes). Depth has a much milder effect on performance, with a flat optimum laying between three and seven layers. However, deeper networks typically maximized performance in fewer epochs than shallow networks did. As in [33], we find that networks perform well at a wide range of width-depth ratios, a finding somewhat inconsistent with [64]. Figure 2 shows that, for four tasks in the Primary Sweep, networks with width-to-depth aspect ratios across many orders of magnitude can achieve good test losses. However, performance degrades for deep and narrow networks and, to a lesser extent, wide and shallow networks. Each data point in this figure depicts the median minimum test loss for a single experiment. Figure 3 visualizes how relative training variation is high surrounding the interpolation threshold: sometimes a training run is “lucky” than others; further study could strengthen our understanding of the LTH.

We observed the greatest differences in generalization error between parameter count and epoch, and see hints of epoch-wise double descent at high parameter counts (see depth 8 in Figure 3 and Appendix B). We were curious to see if epoch-wise double descent could be triggered by either training longer, using larger networks, or adding label noise (as suggested in [24]). So, we ran the 300 Epoch Sweep on larger networks than in the Primary Sweep, the 30k Epoch Sweep on the smaller half of parameter counts from the Primary Sweep, and the Label Noise Sweep repeating the Primary Sweep with increasing levels of label noise. For regression tasks, additive Gaussian noise with a standard deviation equal to a percentage of the standard deviation of the response variable was used in place of label permutation. While these sweeps did not show strong evidence of epoch-wise double descent, they provide a broader view into the impacts of training time and parameter count (see Appendix B Figures 25 and 26). The 30k sweep can act as a validation set when compared to the Primary Sweep, and the 300 sweep extends the primary into larger parameter counts (see Appendix A). As visualized in Figure 4 and Appendix B Figures 25 and 26, high levels of label noise predictably degrade performance.

Next, we were curious to capture how different learning
Fig. 4. Marginal effects plots for batch size, learning rate, and label noise using each parameter’s corresponding sweep. See Figure 1 for a description of the axes, lines, and shaded region.

rates and batch sizes modulate training performance. Figure 4 suggests batch size has a weak effect on generalization performance and training epochs, but larger batch sizes can increase the number of training epochs to minimize test loss. Figure 5 shows that larger batch sizes required fewer optimization steps but smaller batches are more sample-efficient (also see Appendix B Figure 19 and 20), consistent with [65]. Learning rate had a moderate impact on the best test loss achieved, with optimal rates near $10^{-3}$. Appendix B Figures 17-20 further visualize the Learning Rate and Batch Size Sweeps.

Regularization impacts generalization performance, so we performed a “Regularization Sweep” to measure how L1 and L2 regularization mediate training and overtraining. This sweep trains rectangular networks of the Primary Sweep at 14 levels of L1 and L2 kernel regularization each. Figure 6 shows that heavy regularization can increase test losses and sometimes mildly increase training epochs but, as visualized in Appendix B Figures 21-24, it can also delay the onset and severity of overtraining. Regularization did not induce deep double descent in our experiments.

Lastly, to verify that Adam was not biasing the dataset, we conducted an Optimizer Sweep which repeated experiments on rectangular networks from the Primary Sweep using RMSProp and SGD at several batch sizes and learning rates. Figure 6 reveals that choice of optimizer moderately impacts training speed and quality but does not result in wholly different dynamics (see Appendix B Figures 27 and 28). Therefore, we believe patterns seen in this dataset may also manifest when using other optimizers. SGD shows milder decreases in epochs to minimize test loss as parameter count increases beyond the interpolation threshold. This may be due to its fixed step-size; further investigation is required.
Fig. 6. Marginal effects plots of regularization strength using the Regularization Sweep (left) and optimizer choice using the Optimizer Sweep (right).

For the regularization plots we use the unpenalized test loss. “+ M” cases used 0.9 momentum, other cases used none. See Figure 1 for a description of the axes, lines, and shaded region.

B. How we made BUTTER: a high-throughput distributed empirical machine learning framework

As detailed in Appendix D, we collected training and test performance statistics at each training epoch of each repetition as reported by Tensorflow and stored them in the Postgres database from which BUTTER was extracted. 20 statistics were collected at each epoch including training and test loss, accuracy, mean squared error, and KL-divergence. To coordinate these experiments, we developed an object-oriented Python-based framework that supported the simultaneous coordinated execution of the neural network training runs in this dataset across four HPC clusters [49]. We hope that other researchers study and expand this framework to generate additional large empirical machine learning datasets to inform both theoretical and practical research. The framework allows for specifications of computational experiments to be serialized and enqueued in a database-backed task queue which is used to distribute runs to worker nodes which later record the results into the same database. Table V lists the specifications of the systems we utilized.

C. Data publication

The entire 1.7 TB experimental dataset is available from the Open Energy Data Initiative (OEDI) data repository as partitioned parquet files [48]. Each repetition record contains test and training loss, accuracy, mean squared error, mean absolute error, root mean squared error, squared logarithmic error, hinge loss, squared hinge loss, cosine similarity, and Kullback-Leibler divergence measured at each training epoch. We also record the random seed; version numbers of the framework, Python, and Tensorflow; hostname; and operating system used to execute the run. We also provide a smaller summary dataset aggregated over all repetitions of each experiment including statistics such as average, standard deviation, minimum, maximum, and median test and training losses at each epoch.

V. DISCUSSION

We presented BUTTER, a empirical deep learning dataset to further our understanding of deep learning phenomenon, and hopefully to guide theoretical research and inform practical application of neural networks. It is critical to expand empirical study of machine learning beyond this dataset to different types and aspects of neural network architectures including sparse, convolutional, recurrent, and transformer networks. To
facilitate such future studies, we also publish our distributed experimental framework capable of high-throughput machine learning experimentation across multiple systems. Further analysis into learning rate, label noise, and regularization could generate compelling findings and a theoretical explanation for the trends we described (see Appendix A). If such patterns are shown to hold across more datasets and hyperparameters, it may be possible to optimize the energy-intensive process of hyperparameter tuning by leveraging these analytic relationships. We acknowledge that this research contributes to the energy consumption and carbon footprint of deep learning. However, by providing this dataset as a public resource we hope to reduce future energy costs that would have otherwise been incurred by others in recreating these results and we believe this dataset promotes equity by providing this data to those without the resources to generate it.

ACKNOWLEDGEMENTS

This work was authored by the National Renewable Energy Laboratory (NREL), operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. This work was supported by the Laboratory Directed Research and Development (LDRD) Program at NREL. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

The authors declare that they have no conflict of interest.

REFERENCES

[1] W. Chen, X. Gong, Y. Wei, H. Shi, Z. Yan, Y. Yang, and Z. Wang, “Understanding and accelerating neural architecture search with training-free and theory-grounded metrics,” arXiv preprint arXiv:2108.11939, 2021.

[2] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural networks, vol. 2, no. 5, pp. 359–366, 1989.

[3] A. R. Barron, “Universal approximation bounds for superpositions of a sigmoidal function,” IEEE Transactions on Information theory, vol. 39, no. 3, pp. 930–945, 1993.

[4] Y. Zhang, J. Lee, M. Wainwright, and M. I. Jordan, “On the learnability of fully-connected neural networks,” in Artificial Intelligence and Statistics. PMLR, 2017, pp. 83–91.

[5] M. Kohler and S. Langer, “On the rate of convergence of fully connected deep neural network regression estimates,” The Annals of Statistics, vol. 49, no. 4, pp. 2231–2249, 2021.

[6] S. Langer, “Analysis of the rate of convergence of fully connected deep neural network regression estimates with smooth activation function,” Journal of Multivariate Analysis, vol. 182, p. 104695, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0047259X21002761.

[7] A. Jacot, F. Gabriel, and C. Hongler, “Neural Tangent Kernel: Convergence and Generalization in Neural Networks,” Feb. 2020, arXiv:1806.07572 [cs, math, stat]. [Online]. Available: http://arxiv.org/abs/1806.07572.

[8] R. Eldan and O. Shamir, “The power of depth for feedforward neural networks,” in Conference on learning theory. PMLR, 2016, pp. 907–940.

[9] D. Elbrächter, D. Perekrestenko, P. Grohs, and H. Bölcskei, “Deep neural network approximation theory,” IEEE Transactions on Information Theory, vol. 67, no. 5, pp. 2581–2623, 2021.

[10] J. Lu, Z. Shen, H. Yang, and S. Zhang, “Deep network approximation for smooth functions,” SIAM Journal on Mathematical Analysis, vol. 53, no. 5, pp. 5465–5506, 2021.

[11] H. N. Mhaskar and T. Poggio, “Deep vs. shallow networks: An approximation theory perspective,” Analysis and Applications, vol. 14, no. 06, pp. 505–528, 2016. [Online]. Available: https://doi.org/10.1142/S0219530516400042.

[12] D. Yarotsky, “Optimal approximation of continuous functions by very deep relu networks,” in Conference on learning theory. PMLR, 2018, pp. 639–649.

[13] D. Yarotsky and A. Zhevnerchuk, “The phase diagram of approximation rates for deep neural networks,” Advances in neural information processing systems, vol. 33, pp. 13005–13015, 2020.

[14] M. Pezeshki, A. Mitra, Y. Bengio, and G. Lajoie, “Multi-scale feature learning dynamics: Insights for double descent,” arXiv preprint arXiv:2112.03215, 2021.

[15] L. Chen, Y. Min, M. Belkin, and A. Karbasi, “Multiple descent: Design your own generalization curve,” Advances in Neural Information Processing Systems, vol. 34, 2021.

[16] B. Adlam and J. Pennington, “The neural tangent kernel in high dimensions: Triple descent and a multi-scale theory of generalization,” in Proceedings of the 37th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 74–84. [Online]. Available: https://proceedings.mlr.press/v119/adlam20a.html.

[17] S. Geman, E. Bienenstock, and R. Doursat, “Neural networks and the bias/variance dilemma,” Neural computation, vol. 4, no. 1, pp. 1–58, 1992.

[18] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, The elements of statistical learning: data mining, inference, and prediction. Springer, 2009, vol. 2.

[19] B. Neyshabur, R. Tomioka, and N. Srebro, “In search of the real inductive bias: On the role of implicit regularization in deep learning,” in ICML (Workshop), 2015.

[20] B. Neyshabur, “Implicit regularization in deep learning,” arXiv preprint arXiv:1709.01953, 2017.

[21] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scichta, S. Lacoste-Julien, and I. Mitliagkas, “A modern take on the bias-variance tradeoff in neural networks,” arXiv preprint arXiv:1810.08591, 2018.

[22] M. Belkin, D. Hsu, S. Ma, and S. Mandal, “Reconciling modern machine-learning practice and the classical bias\&#x2122;variance trade-off,” Proceedings of the National Academy of Sciences, vol. 116, no. 32, pp. 15 849–15 854, 2019. [Online]. Available: https://www.pnas.org/doi/10.1073/pnas.1903070116.

[23] P. Nakkiran, G. Kaplan, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, “Deep Double Descent: Where Bigger Models and More Data Hurt,” International Conference on Learning Representations, Dec. 2019. [Online]. Available: http://arxiv.org/abs/1912.02292.

[24] ——, “Deep double descent: Where bigger models and more data hurt,” Journal of Statistical Mechanics: Theory and Experiment, vol. 2021, no. 12, p. 124003, 2021.

[25] A. Klein and F. Hutter, “Tabular benchmarks for joint architecture and hyperparameter optimization,” arXiv preprint arXiv:1905.04970, 2019.

[26] Y. Hirose, N. Yoshinari, and S. Shirakawa, “Nas-hpo-bench-ii: A benchmark dataset on joint optimization of convolutional neural network architecture and training hyperparameters,” in Asian Conference on Machine Learning. PMLR, 2021, pp. 1349–1364.

[27] L. Zimmer, M. Lindauer, and F. Hutter, “Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 9, pp. 3079–3090, 2021.

[28] X. Dong and Y. Yang, “Nas-bench-201: Extending the scope of reproducible neural architecture search,” in International Conference on Learning Representations, 2020. [Online]. Available: https://openreview.net/forum?id=HlxyZkBKDr.

[29] X. Dong, L. Liu, K. Masial, and B. Gabrys, “Nats-bench: Benchmarking nas algorithms for architecture topology and size,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, pp. 3634–3646, 2022.

[30] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, “Nas-bench-101: Towards reproducible neural architecture search,” in International Conference on Machine Learning. PMLR, 2019, pp. 7105–7114.

[31] A. Mehrotra, A. G. C. P. Ramos, S. Bhattacharya, L. Dudziak, R. Vipperla, T. Chau, M. S. Abdelfattah, S. Ishihaq, and N. D. Lane, “NAS-bench-(asr): Reproducible neural architecture search for speech recognition,” in
Charles Edison Tripp (Member, IEEE) received the B.S. in Electrical Engineering from Rice University in 2006, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University in 2008 and 2013 respectively. He is a senior scientist in the Artificial Intelligence, Learning, and Intelligent Systems group of the Computational Science Center at the National Renewable Energy Laboratory in Golden, Colorado, USA. His research interests include machine learning, reinforcement learning, Bayesian methods, and decision science.

Jordan Perr-Sauer received his B.S. in Applied Mathematics from University of Colorado, and his M.S. in Computer Science from the University of Colorado, Boulder in 2022. He is a data science researcher in the Data, Analysis, and Visualization group of the Computational Science Center at the National Renewable Energy Laboratory in Golden, Colorado, USA. His research interests include explainable and interpretable machine learning and software engineering for science.

Lucas Hayne (Member, IEEE) received a B.S. in Computer Science and Psychology in 2017, an M.S. in Computer Science in 2021, and plans on graduating with his Ph.D. in 2023 from the University of Colorado Boulder. He is a Data Analyst in the Artificial Intelligence, Learning, and Intelligent Systems group of the Computational Science Center at the National Renewable Energy Laboratory in Golden, Colorado, USA. His research interests include machine learning interpretability and cognitive science.

Monte Lunacek received his B.S. in Applied Mathematics from the University of Colorado, Colorado Springs, and his M.S. and Ph.D. degrees in Computer Science from Colorado State University. He is a Senior Scientist in the Computational Science Center at the National Renewable Energy Laboratory. His research interests include distributed computing, machine learning, data optimization and analysis, and web visualization.

Jamil Gafur received his B.S. in Computer Science from CUNY Lehman College in 2018. Currently, a Ph.D. Student in Computer Science at the University of Iowa working on localized explanations of neural networks through swarm algorithms. He is a Computational Scientist in the Artificial Intelligence, Learning, and Intelligent Systems group of the Computational Science Center at the National Renewable Energy Laboratory in Golden, Colorado, USA. His research interests include adversarial attacks, machine learning, and heuristic optimization.
APPENDIX

This Appendix contains additional details and links that can be used to access and reproduce the BUTTER dataset, as well as a number of plots visualizing several dimensions of interest within the dataset. Each plot visualizes attributes from each sweep of the dataset. In each plot, we vary one attribute at a time while other attributes remain constant. **Unless otherwise stated we use the following as default values for each attribute:** shape: rectangle, depth: 3, learning rate: 0.0001, batch size: 256, label noise: 0.0, and regularization penalty: 0.0. For all plots depicting median losses, we restrict the contours to losses less than 10 times the minimum loss unless otherwise stated. Triangles mark the minimum median test losses. A gray line marks the median epoch at which test loss was minimized across all repetitions of that experiment. Colors are logarithmically scaled to show detail and are normalized by task. To reveal details in regions with less variation in plots depicting coefficients of variation, we offset the interquartile range / median slightly before scaling.

Appendix Contents:

- Appendix A: A Preliminary Analysis of The Dataset
- Appendix B: Sweep Overview Plots
  - Figures 9-12: Network Shape Plots
  - Figures 13-16: Network Depth Plots
  - Figures 17-18: Learning Rate Plots
  - Figures 19-20: Minibatch Size Plots
  - Figures 21-22: $L_1$ Regularization Plots
  - Figures 23-24: $L_2$ Regularization Plots
  - Figures 25-26: Label Noise Plots
  - Figures 27-28: Optimizer Comparison Plots
- Appendix C: Dataset Information
- Appendix D: Dataset Datasheet
A. Preliminary analysis

Here we present a preliminary analysis of patterns in the test loss observed across # parameters, training epoch, learning task, and network shape. The purpose of this analysis is to demonstrate some consistent patterns that we observe in the data, and evaluate how well those patterns generalize to data with larger numbers of parameters and epochs. Such a result could strengthen our understanding of deep learning and inform the hyperparameter tuning process. This is only a peek at what future analysis could illuminate, and uses only a fraction of the data contained within the dataset.

Figure 7 provides contour plot visualization of the generalization statistics of the Primary Sweep, we sought to test our fit functions on data from the Primary Sweep. The contour plots were created using the tricontourf function of Matplotlib utilizing the Delaunay triangulation method [66]. Faint vertical banding or jagged contours in this figure may be artifacts of the visualization technique due to the regular spacing of data along the horizontal axis. Each data point plotted is aggregated over at least twenty repetitions of that experiment with distinct random seeds. In these plots we observed a similar curvilinear shape of parameter counts with minimums above this threshold. These generalization performance relationships in each plot shades parameter counts with minimums above the log of the parameter count at which it occurs.

1) Fitting and extrapolating analytic curves: In this section, we seek to establish a relationship between the number of parameters, the smallest loss it can achieve, the number of epochs to do so, and how rapidly it descends to that minimum. Specifically, we consider the # of parameters, the median test loss, the epoch at which the median test loss is minimized, and the log-space slope measured between the minimum and the earliest epoch to achieve 10% greater test loss of the minimum itself. To illustrate our analysis, we examine 4 shapes and 7 tasks with a depth (number of hidden layers) of 3 from the Primary Sweep and smooth their per-epoch median test loss curves using a linear Savitzky-Golay filter with a window size of 101 [67]. The minimizing epoch, minimum median test loss, and descent slope of the smoothed curves are plotted in Figure 7.

Next, we identified analytic functions which are consistent with this data and can be used to estimate these three values for any parameter count on a given learning task,

\[ \theta_{min}^{-1/3} = m \log(\sigma) + b \]  
\[ \log(\phi_{min}) = c(\log(\sigma) - s_0)^{-p} + \phi \]  
\[ \alpha = \frac{\Delta \log(\phi)}{\Delta \log(\theta)} = -l(\log(\sigma) - s_1)^2 + \alpha_0 \]

where \( \theta \) is the epoch; \( \theta_{min} \) is the minimizing epoch; \( \sigma \) is the number of trainable parameters in the network; \( \phi \) is the median test loss; \( \phi_{min} \) is the minimum median test loss; \( \alpha \) is the log space descent slope approaching \( \theta_{min} \); and \( m, b, c, p, \phi, s_0, s_1, \text{ and } \alpha_0 \) are constant for each task and shape. Equation 1 estimates that the logarithm of the number of trainable parameters is approximately linearly proportional to the inverse cube root of the minimizing epoch. Equation 2 suggests that log of the minimum test loss has an approximate power law relationship to the log of the parameter count with exponent \(-p\), scale \( c \), and offset \( \phi \). \( s_0 \) is the log of the minimum number of parameters needed to learn the task, and \( \phi \) is the log of the minimum achievable test loss. Finally, Equation 3 suggests the log slope of test loss as it approaches \( \theta_{min} \) is approximately quadratically related to the log of the parameter count where \( l \) is a scale factor, and \( \alpha_0 \) and \( s_1 \) are the most gentle slope and the log of the parameter count at which it occurs.

Figure 7 shows non-linear least squares fits as generated with SciPy’s least_squares solver [68] of Equations 1, 2, and 3 to the rectangular network data points. To prevent fitting to experiments which might minimize test loss beyond the tested 3000 epochs, we only fit Equation 1 to parameter counts with measured minimums before the 2900th epoch. The grey region in each plot shades parameter counts with minimums above this threshold. These generalization performance relationships are similar across all of the network shapes we examined.

After fitting functions to the generalization statistics of the Primary Sweep, we sought to test our fit functions on data beyond that used to identify these relationships. Figure 8 shows the outcome of visualizing those tests on the connect_4 task, with the black dashed line indicating the fit of Equation 1 and 2 to the Primary Sweep data from Figure 7 overlaid on the corresponding 30k Epoch Sweep and 300 Epoch Sweep heatmaps. Equations 1 and 2 provide promisingly reasonable estimates of minimum test loss locations for networks trained in these additional sweeps. This visualization serves as a first step in validating Equations 1 and 2 as general trends in the deep learning process. The preliminary analysis presented here, while promising, only utilizes a small fraction of the dataset and the basis for these relationships is solely empirical.

For networks trained on the 529_pollen task we used a window size of 901 and second order polynomial filter to stabilize noisy minimum test losses observed in that task.
Fig. 7. Generalization statistics from the Primary Sweep. Each row depicts statistics for a given task and each column a different generalization statistic. For each # of trainable parameters, the first column plots the number of epochs needed to achieve the minimum test loss, the second column plots the minimum test loss, and the third column the descent slope towards that minimum. The dashed line plots Equations 1, 2, and 3 fit to the rectangular network data points.
Fig. 8. Predictions of generalization statistics in 300 and 30k Epoch sweeps. The top half of the figure shows log-scaled data from three sweeps for a rectangular network trained on connect_4. The black dotted line depicts Equation 1 fit to Primary Sweep data. Three different icons indicate the locations of test loss minimums in each sweep. The bottom plot shows measured test loss minimums for each sweep where the dotted black line represents Equation 2 fit to only the Primary Sweep.
B. Sweep Overview Plots

- Figures 9-12: Network Shape Plots
- Figures 13-16: Network Depth Plots
- Figures 17, 18: Learning Rate Plots
- Figures 19, 20: Minibatch Size Plots
- Figures 21, 22: $L^1$ Regularization Plots
- Figures 23, 24: $L^2$ Regularization Plots
- Figures 25, 26: Label Noise Plots
- Figures 27, 28: Optimizer Comparison Plots
Fig. 9. Median test loss for tasks and the first half of shapes in the Primary Sweep. Patterns are fairly consistent across shapes. The "butter-zone" of low median test loss forms a curvilinear shape in log-space for all tasks.
Fig. 10. Interquartile range / median for tasks and the first half of shapes in the Primary Sweep. Variation decreased in the butter-zone with higher spread in regions of over- and under-fitting. Small parameter counts generally had greater variation in test loss. This pattern may support the lottery ticket hypothesis which predicts that increasing the number of parameters in a network increases the probability that gradient descent converges.
Fig. 11. Median test loss for tasks and the second half of shapes in the Primary Sweep. Similar patterns emerge as in Figure 9.
Fig. 12. Interquartile range / median for tasks and the second half of shapes in the Primary Sweep. Similar patterns emerge as in Figure 10.
Fig. 13. Median test loss for tasks and the first half of depths in the Primary Sweep. The depth at which the lowest median test loss was achieved varied by task. For example, very shallow networks trained on mnist generalized best, but the reverse is true for 201_pol where deeper networks generalized better.
Fig. 14. Interquartile range / median for tasks and the first half of depths in the Primary Sweep. Variations in test loss generally increased with increased depth. Particularly, the vertical orange band of high spread that covers small networks tends to shift rightward with depth suggesting that more parameters may be needed to stabilize training at higher depths. Additionally, variability in overtrained regions increased more for deeper networks.
Fig. 15. Median test loss for tasks and the second half of depths in the Primary Sweep. Depth two is also shown for comparison. Similar patterns emerge as in Figure 13.
Fig. 16. Interquartile range / median for tasks and the second half of depths in the Primary Sweep. Depth two is also shown for comparison. Similar patterns emerge as in Figure 14.
Fig. 17. Median test loss for tasks and learning rates in the Learning Rate Sweep. Higher learning rate networks decreased median test loss more rapidly early on, but sometimes did not achieve as low of a minimum median test loss. This can be seen in the case of connect_4 and mnist, where fast learning rates coincided with high minimum losses when compared to those achieved using slower rates. Additionally, larger networks showed more sensitivity to the learning rate than did smaller networks. We restrict the contours to losses less than 2 times the minimum loss.
Fig. 18. Interquartile range / median for tasks and learning rates in the Learning Rate Sweep. Higher learning rates coincided with elevated variation across experiment repetitions. This is especially apparent when networks were overtrained (larger parameter counts and higher epochs). For some tasks (e.g., 529_pollen and 537_houses) the highest learning rate prevented training altogether at the largest parameter counts where variations between repetitions become very large.
Fig. 19. **Median test loss for tasks and batch sizes in the Batch Size Sweep.** For all tasks shown here, increasing the batch size coincided with an increase in the number of epochs taken to minimize test loss. In fact, the entire "butter-zone" tends to shift upwards as the batch size increases. We restrict the contours to losses less than 2 times the minimum loss.
Fig. 20. Interquartile range / median for tasks and batch sizes in the Batch Size Sweep. Because of the upward butter-zone shifts observed when increasing the batch size, small batch size plots show more overtraining variation and large batch size plots show more undertraining variation.
Fig. 21. Median test loss for tasks and $L^1$ regularization penalty levels in the Regularization Sweep. Increasing the $L^1$ regularization penalty generally increased the minimum test loss achieved, and increased the number of epochs to achieve a target test loss. High penalties coincided with high test losses in large networks trained for small numbers of epochs. $L^1$ regularization reduced or eliminated overtraining in the experiments visualized here. In a few cases, regularization seems to ‘bend’ the butter-zone upwards for higher parameter counts, delaying learning. More investigation is warranted to understand this effect.
Fig. 22. Interquartile range / median for tasks and $L^1$ regularization penalties in the Regularization Sweep. Larger $L^1$ penalties coincided with less variation between repetitions.
Fig. 23. Median test loss for tasks and $L^2$ regularization penalty levels in the Regularization Sweep. Increasing the $L^2$ regularization penalty generally increased the minimum test loss achieved, and increased the number of epochs to achieve a target test loss. Generalization was poor when high penalties were applied on large networks trained for small numbers of epochs. However, $L^2$ regularization reduced or eliminated overtraining in the experiments visualized here. Similarly to Figure 21, this figure is disproportionately affected by the addition of a regularization penalty to the test loss.
Fig. 24. Interquartile range / median for tasks and $L^2$ regularization penalties in the Regularization Sweep. Increased $L^2$ penalties decreased the variation between repetitions.
Fig. 25. **Median test loss for tasks and label noise levels in the Label Noise Sweep.** Increased label noise levels affected each task in a different way. Increasing the label noise for mnist, even by 5% has a large adverse impact on training. However, larger amounts of label noise have little or no effect on test loss for the 529_pollen task. We restrict the contours to losses less than 2 times the minimum loss.
Fig. 26. Interquartile range / median for tasks and label noise levels in the Label Noise Sweep. High label noise levels produced high variations in test loss of overtrained networks.
Fig. 27. Median test loss for tasks and optimizers in the Optimizer Sweep. RMSProp and SGD both perform similarly to Adam in these experiments. Possibly due to a static step-size, SGD has a “flatter” butter-zone beyond the interpolation threshold. For SGD we plotted experiments with a learning rate of 0.01 instead of 0.001.
Fig. 28. **Interquartile range / median for tasks and optimizers in the Optimizer Sweep.** We see similar bands of variation before (strongly) and after (diffusely) the bottom of the lowest test loss region. Regardless of optimizer, networks sometimes learn faster and sometimes learn slower, particularly near the interpolation threshold.
C. Dataset Information

The BUTTER dataset is intended to provide empirical deep learning performance data to inform machine learning researchers and practitioners; to act as a guiding light for theoreticians and engineers in investigating, understanding, and furthering our knowledge of the deep learning phenomenon.

The BUTTER dataset is permanently publicly available through the Open Energy Data Initiative (OEDI) under the Creative Commons Attribution-ShareAlike 4.0 International Public License. The entire dataset can be accessed through the OEDI S3 viewer. A metadata description of the storage schema and exact dataset contents is linked from the OEDI dataset page, and can be accessed directly from the OEDI GitHub repository. Example code that generates figures from this paper using the dataset can be found in the BUTTER visualization repository.

As described in Table II, the dataset is composed of 7 hyperparameter sweeps each scanning across a grid of experimental parameters. Table III enumerates the network shapes and IV enumerates the tasks referenced in Table II. The number of repetitions listed in Table II is typical, however due to overlapping sweeps, in some instances more repetitions are stored in the dataset. For example, the Learning Rate Sweep has 20 typical repetitions, but in experiments shared with the Primary Sweep, 30 repetitions are recorded. Additionally, a small number of experiments in a sweep have fewer than the typical number of repetitions. The exact number of repetitions of each experiment is stored in the summary dataset, and can be calculated from the full dataset by aggregating repetitions over experiment_id.

The BUTTER dataset was accumulated from 2021 to 2022 utilizing the four computing systems listed in Table V. The software used to accumulate the dataset, including complete commit history is available under the MIT license. For traceability and reproducibility, each repetition record in the dataset records the start and end time of the training run, along with the hostname, operating system, software version, git hash, Python version, and Tensorflow version used to execute it.

The BUTTER dataset does not contain any offensive or personally identifiable information. While we have taken measures to mitigate the risk, the dataset may contain incorrect or corrupted data. The authors bear all responsibility in case of violation of rights.

D. Dataset Datasheet

This section contains a dataset datasheet as described in [69].

1) Motivation: For what purpose was the dataset created?

The BUTTER dataset is intended to provide empirical deep learning performance data to inform machine learning researchers and practitioners; to act as a guiding light for theoreticians and engineers in investigating, understanding, and furthering our knowledge of the deep learning phenomenon. We were specifically motivated by the desire to construct evidence-informed priors for efficient hyperparameter search and selection.

Who created the dataset?

The dataset was created by Dr. Charles Tripp, Jordan Perr-Sauer, Lucas Hayne, and Dr. Monte Lunacek as part of an internally funded machine learning project within the Computational Science Center at the National Renewable Energy Laboratory (NREL), a United States Department of Energy (DOE) National Laboratory operated by the Alliance for Sustainable Energy, LLC.

Who funded the dataset?

Creation of the dataset was funded using NREL laboratory directed research and development funds.

2) Composition: What do the instances that comprise the dataset represent?

As described in the dataset readme, the raw dataset contains records of deep learning training runs, repeating each distinct experiment multiple times using different random seeds.

Each row of the full dataset represents a single repetition of an experiment (that is, a single training run). Each repetition was instrumented to record various statistics at the end of each training epoch, and those statistics are stored as epoch-indexed arrays in each row. Runs are labeled with an 'experiment_id' which was used to aggregate repetitions of the same experimental parameters together in the summary dataset. An experiment_id in the summary dataset correspond to the same experiment_id in the summary dataset.

For convenience, a summary dataset is also provided which contains statistics of these measurements aggregated over every repetition of each distinct experiment.

How many instances are there in total?

The full dataset contains 11.2 million total training runs, and the summary dataset contains 483 thousand distinct experiment records covering 40 billion training epochs.

Does the dataset contain all possible instances or is it a sample of instances from a larger set?

The dataset contains all of the training runs we executed.

What data does each instance consist of?

As described in the dataset readme, the complete raw dataset is available in the /all_runs/ partitioned parquet dataset. Each row in this dataset is a record of one training repetition of a network. Several statistics were recorded at the end of each training epoch, and those records are stored in this row as arrays indexed by training epoch. For convenience, we also provide the /complete_summary/ partitioned parquet dataset which contains statistics aggregated over all repetitions of the same experiment including average and median test and training losses at the end of each training epoch. Distinct experiments are uniquely and consistently identified in both datasets by an 'experiment_id'. Additionally, we have created separate full (containing all
repetitions) and per-experiment summary datasets for each experimental sweep so that they can be downloaded and queried separately if the entire dataset is not needed. The schemas of summary and full datasets are the same for every sweep.

b) File Hierarchy and Descriptions:
- `/complete_executive_summary/` This file is intended to provide a simple, small dataset that can be more easily and quickly downloaded, queried, and analyzed than the summary or run datasets and provides an easy starting point for using this dataset. It contains a minimal set of per-experiment statistics aggregated over every repetition of each distinct experiment for all sweeps. This file has the same schema as the summary datasets, except it does not include any per-epoch statistic columns except for test_loss_q1, test_loss_median, and test_loss_q3.
- `/complete_executive_summary.tar.xz` is a compressed tarball of `/complete_executive_summary/`
- `/complete_summary/` contains per-experiment statistics aggregated over every repetition of each distinct experiment for all sweeps. This complete summary dataset is designed for easy analysis of aggregate statistics without the need to query individual repetitions. If the executive summary doesn’t have everything you need (for example you want to see the KL-Divergence history for regularized experiments), the summary dataset likely is what you want to use.
- `/complete_summary.tar.xz` is a compressed tarball of `/complete_summary/`
- `*/all_repetitions/**` contains all of the repetition records in all sweeps. This dataset contains all of the raw data logged during every repetition of every experiment. It is well-partitioned, but even so, it can be cumbersome to work with. Use the executive summary or summary datasets if you can. If you need the raw data, though, it’s in here.
- `/all_repetitions.tar` is a tarball of `/all_repetitions/`
- `/primary_summary/` contains summary experiment statistics for the Primary Sweep
- `/primary_summary.tar.xz` is a compressed tarball of `/primary_summary/`
- `/300_epoch_summary/` contains summary experiment statistics for the 300 epoch sweep
- `/300_epoch_summary.tar.xz` is a compressed tarball of `/300_epoch_summary/`
- `/30k_epoch_summary/` contains summary experiment statistics for the 30k epoch sweep
- `/30k_epoch_summary.tar.xz` is a compressed tarball of `/30k_epoch_summary/`
- `/learning_rate_summary/` contains summary experiment statistics for the learning rate sweep
- `/learning_rate_summary.tar.xz` is a compressed tarball of `/learning_rate_summary/`
- `/label_noise_summary/` contains summary experiment statistics for the label noise sweep
- `/label_noise_summary.tar.xz` is a compressed tarball of `/label_noise_summary/`
- `/batch_size_summary/` contains summary experiment statistics for the batch size sweep
- `/batch_size_summary.tar.xz` is a compressed tarball of `/batch_size_summary/`
- `/regularization_summary/` contains summary experiment statistics for the regularization sweep
- `/regularization_summary.tar.xz` is a compressed tarball of `/regularization_summary/`
- `/optimizer_summary/` contains summary experiment statistics for the optimizer sweep
- `/optimizer_summary.tar.xz` is a compressed tarball of `/optimizer_summary/`

c) Experiment Summary Schema: For preliminary analysis, we recommend using the summary dataset as it is smaller and more convenient to work with than the full repetition dataset. However, the entire record of every repetition of every experiment is stored in the full dataset, allowing other statistics to be computed from the raw data. Each experiment has a unique experiment_id value, which matches repetition records in the runs dataset. Summary data is partitioned by dataset, shape, learning rate, batch size, kernel regularizer, label noise, depth, and number of training epochs.

Columns Describing Each Experiment
- experiment_id: the unique id for this experiment
- primary_sweep: bool, true iff this experiment is part of the Primary Sweep
- 300_epoch_sweep: bool, true iff this experiment is part of the 300 epoch sweep
- 30k_epoch_sweep: bool, true iff this experiment is part of the 30k epoch sweep
- learning_rate_sweep: bool, true iff this experiment is part of the learning rate sweep
- label_noise_sweep: bool, true iff this experiment is part of the label noise sweep
- batch_size_sweep: bool, true iff this experiment is part of the batch size sweep
- regularization_sweep: bool, true iff this experiment is part of the regularization sweep
- optimizer_sweep: bool, true iff this experiment is part of the optimizer sweep
- activation: string, the activation function used for hidden layers
- batch: string, a nickname for the experimental batch this experiment belongs to
- batch_size: uint32, minibatch size
- dataset: string, name of the dataset used
- depth: uint8, number of layers
- early_stopping: string, early stopping policy
- epochs: uint32, number of training epochs in this experiment
- input_activation: string, input activation function
- kernel_regularizer: string, null if no regularizer is used
- kernel_regularizer_l1: float32, L1 regularization penalty coefficient
- kernel_regularizer_l2: float32, L2 regularization penalty coefficient
- kernel_regularizer_type: string, name of kernel regularizer used (null if none used)
- label_noise: float32, amount of label noise applied to
Experiment:

Columns Describing Overall Training Trajectories for Each Experiment:

- num: [uint8], number of runs aggregated in this summary record at each epoch

- Test Loss Trajectory Statistics:
  - test_loss_num_finite: [uint8], number of finite test losses at each epoch
  - test_loss_min: [float32], minimum test loss at each epoch
  - test_loss_q1: [float32], first quartile test loss at each epoch
  - test_loss_median: [float32], median test loss at each epoch
  - test_loss_q3: [float32], third quartile test loss at each epoch
  - test_loss_max: [float32], maximum test loss at each epoch
  - test_loss_avg: [float32], average test loss at each epoch
  - test_loss_stddev: [float32], standard deviation of the test loss at each epoch

- Training Loss Trajectory Statistics:
  - train_loss_num_finite: [uint8], number of finite training losses at each epoch
  - train_loss_min: [float32], minimum training loss at each epoch
  - train_loss_q1: [float32], first quartile training loss at each epoch
  - train_loss_median: [float32], median training loss at each epoch
  - train_loss_q3: [float32], third quartile training loss at each epoch
  - train_loss_max: [float32], maximum training loss at each epoch
  - train_loss_avg: [float32], average training loss at each epoch
  - train_loss_stddev: [float32], standard deviation of training losses at each epoch

- Test Accuracy Trajectory Statistics:
  - test_accuracy_num_finite: [uint8], number of finite test accuracies at each epoch
  - test_accuracy_min: [float32], minimum test accuracy at each epoch
  - test_accuracy_q1: [float32], first quartile test accuracy at each epoch
  - test_accuracy_median: [float32], median test accuracy at each epoch
  - test_accuracy_q3: [float32], third quartile test accuracy at each epoch
  - test_accuracy_max: [float32], maximum test accuracy at each epoch
  - test_accuracy_avg: [float32], average test accuracy at each epoch
  - test_accuracy_stddev: [float32], standard deviation of test accuracy at each epoch

- Training Accuracy Trajectory Statistics:
  - train_accuracy_num_finite: [uint8], number of finite training accuracies at each epoch
  - train_accuracy_min: [float32], minimum training accuracy at each epoch
  - train_accuracy_q1: [float32], first quartile training accuracy at each epoch
  - train_accuracy_median: [float32], median training accuracy at each epoch
  - train_accuracy_q3: [float32], third quartile training accuracy at each epoch
  - train_accuracy_max: [float32], maximum training accuracy at each epoch
  - train_accuracy_avg: [float32], average training accuracy at each epoch
  - train_accuracy_stddev: [float32], standard deviation of training accuracy at each epoch

- Test Mean Squared Error (MSE) Trajectory Statistics:
  - test_mean_squared_error_num_finite: [uint8], number of finite test MSEs at each epoch
  - test_mean_squared_error_min: [float32], minimum test MSE at each epoch
  - test_mean_squared_error_q1: [float32], first quartile test MSE at each epoch
  - test_mean_squared_error_median: [float32], median test MSE at each epoch
  - test_mean_squared_error_q3: [float32], third quartile test MSE at each epoch
  - test_mean_squared_error_max: [float32], maximum test MSE at each epoch
  - test_mean_squared_error_avg: [float32], average test MSE at each epoch
  - test_mean_squared_error_stddev: [float32], standard deviation of test MSE at each epoch

- Training Mean Squared Error (MSE) Trajectory Statistics:
  - train_mean_squared_error_num_finite: [uint8], number of finite training MSEs at each epoch
  - train_mean_squared_error_min: [float32], minimum training MSE at each epoch
  - train_mean_squared_error_q1: [float32], first quartile training MSE at each epoch
  - train_mean_squared_error_median: [float32], median training MSE at each epoch
  - train_mean_squared_error_q3: [float32], third quartile training MSE at each epoch
  - train_mean_squared_error_max: [float32], maximum training MSE at each epoch
  - train_mean_squared_error_avg: [float32], average training MSE at each epoch
  - train_mean_squared_error_stddev: [float32], training MSE standard deviation at each epoch

- Test Kullback-Leibler Divergence (KL-Divergence) Trajectory Statistics:
  - test_kullback_leibler_divergence_num_finite: [uint8], number of finite test KL-Divergences at each epoch
  - test_kullback_leibler_divergence_min: [float32], minimum test KL-Divergence at each epoch
  - test_kullback_leibler_divergence_q1: [float32], first quartile test KL-Divergence at each epoch
  - test_kullback_leibler_divergence_median: [float32], median test KL-Divergence at each epoch
  - test_kullback_leibler_divergence_q3: [float32], third quartile test KL-Divergence at each epoch
  - test_kullback_leibler_divergence_max: [float32], maximum test KL-Divergence at each epoch
  - test_kullback_leibler_divergence_avg: [float32], average test KL-Divergence at each epoch
  - test_kullback_leibler_divergence_stddev: [float32], test KL-Divergence standard deviation at each epoch

- Training Kullback-Leibler Divergence (KL-Divergence) Trajectory Statistics:
- `train_kullback_leibler_divergence_q1`: [float32], first quartile training KL-Divergence at each epoch
- `train_kullback_leibler_divergence_median`: [float32], median training KL-Divergence at each epoch
- `train_kullback_leibler_divergence_q3`: [float32], third quartile training KL-Divergence at each epoch
- `train_kullback_leibler_divergence_avg`: [float32], average training KL-Divergence at each epoch
- `train_kullback_leibler_divergence_stddev`: [float32], training KL-Divergence standard deviation at each epoch

Columns Describing Performance at the Epoch that Minimized a Statistic (e.g.: the epoch that achieved the lowest test loss)
- Statistics about Per-Repetition Points of Minimum Test Loss
  - Distribution of the Epoch of Minimum Test Loss of each Repetition:
    - `test_loss_min_epoch_min`: [float32], earliest epoch at which test loss was minimized among all repetitions of the experiment
    - `test_loss_min_epoch_q1`: [float32], first quartile of the epoch at which test loss was minimized
    - `test_loss_min_epoch_median`: [float32], median epoch at which test loss was minimized
    - `test_loss_min_epoch_q3`: [float32], third quartile of the epoch at which test loss was minimized
    - `test_loss_min_epoch_max`: [float32], latest epoch at which test loss was minimized among all repetitions of the experiment
    - `test_loss_min_epoch_avg`: [float32], average epoch at which test loss was minimized

- Distribution of the Value of the Minimum Test Loss of each Repetition:
  - `test_loss_min_value_min`: [float32], lowest minimum test loss
  - `test_loss_min_value_q1`: [float32], first quartile minimum test loss
  - `test_loss_min_value_median`: [float32], median minimum test loss
  - `test_loss_min_value_q3`: [float32], third quartile minimum test loss
  - `test_loss_min_value_max`: [float32], highest minimum test loss
  - `test_loss_min_value_avg`: [float32], average minimum test loss

- Statistics about Per-Repetition Points of Maximum Test Accuracy:
  - Distribution of the Epoch of Maximum Test Accuracy of each Repetition:
    - `test_accuracy_max_epoch_min`: [float32], earliest epoch at which test loss was minimized among all repetitions of the experiment
    - `test_accuracy_max_epoch_q1`: [float32], first quartile of the epoch at which test loss was minimized
    - `test_accuracy_max_epoch_median`: [float32], median epoch at which test loss was minimized
    - `test_accuracy_max_epoch_q3`: [float32], third quartile of the epoch at which test loss was minimized
    - `test_accuracy_max_epoch_max`: [float32], latest epoch at which test loss was minimized among all repetitions of the experiment
    - `test_accuracy_max_epoch_avg`: [float32], average epoch at which test loss was minimized

- Distribution of the Value of the Maximum Test Accuracy of each Repetition:
  - `test_accuracy_max_value_min`: [float32], lowest minimum test loss
  - `test_accuracy_max_value_q1`: [float32], first quartile minimum test loss
  - `test_accuracy_max_value_median`: [float32], median minimum test loss
  - `test_accuracy_max_value_q3`: [float32], third quartile minimum test loss
  - `test_accuracy_max_value_max`: [float32], highest minimum test loss
  - `test_accuracy_max_value_avg`: [float32], average minimum test loss

- Statistics about Per-Repetition Points of Minimum Test MSE:
  - Distribution of the Epoch of Minimum Test MSE of each Repetition:
    - `test_mean_squared_error_min_epoch_min`: [float32], earliest epoch at which test MSE was minimized among all repetitions of the experiment
    - `test_mean_squared_error_min_epoch_q1`: [float32], first quartile of the epoch at which test MSE was minimized
    - `test_mean_squared_error_min_epoch_median`: [float32], median epoch at which test MSE was minimized
    - `test_mean_squared_error_min_epoch_q3`: [float32], third quartile of the epoch at which test MSE was minimized
    - `test_mean_squared_error_min_epoch_max`: [float32], latest epoch at which test MSE was minimized among all repetitions of the experiment
    - `test_mean_squared_error_min_epoch_avg`: [float32], average epoch at which test MSE was minimized

- Distribution of the Value of the Minimum MSE of each Repetition:
  - `test_mean_squared_error_min_value_min`: [float32], lowest minimum test MSE
  - `test_mean_squared_error_min_value_q1`: [float32], first quartile minimum test MSE
  - `test_mean_squared_error_min_value_median`: [float32], median minimum test MSE
  - `test_mean_squared_error_min_value_q3`: [float32], third quartile minimum test MSE
  - `test_mean_squared_error_min_value_max`: [float32], highest minimum test MSE
  - `test_mean_squared_error_min_value_avg`: [float32], average minimum test MSE
was used to aggregate repetitions of the same experimental
in the summary dataset. The columns of the full dataset are:

- experiment_id: uint32, id of the experiment this run was
  a repetition of
- run_id: string, unique id of this repetition
- primary_sweep: bool, true iff this experiment is part of
  the Primary Sweep
- 300_epoch_sweep: bool, true iff this experiment is part
  of the 300 epoch sweep
- 30k_epoch_sweep: bool, true iff this experiment is part
  of the 30k epoch sweep
- learning_rate_sweep: bool, true iff this experiment is part
  of the learning rate sweep
- label_noise_sweep: bool, true iff this experiment is part
  of the label noise sweep
- batch_size_sweep: bool, true iff this experiment is part
  of the batch size sweep
- regularization_sweep: bool, true iff this experiment is part
  of the regularization sweep
- optimizer_sweep: bool, true iff this experiment is part of
  the optimizer sweep
- activation: string, the activation function used for hidden
  layers
- batch: string, a nickname for the experimental batch this
  experiment belongs to
- batch_size: uint32, minibatch size
- dataset: string, name of the dataset used
- depth: uint8, number of layers
- early_stopping: string, early stopping policy
- epochs: uint32, number of training epochs in this repetition
- input_activation: string, input activation function
- kernel_regularizer: string, null if no regularizer is used
- kernel_regularizer_l1: float32, L1 regularization penalty
  coefficient
- kernel_regularizer_l2: float32, L2 regularization penalty
  coefficient
- kernel_regularizer_type: string, name of kernel regularizer
  used (null if none used)
- label_noise: float32, amount of label noise applied to
dataset before training (.05 means 5
- learning_rate: float32, learning rate used
- optimizer: string, name of the optimizer used
- output_activation: string, activation function for output
  layer
- python_version: string, python version used
- shape: string, network shape
- size: uint64, approximate number of trainable parameters
  used
- task: string, name of training task
- task_version: uint16, major version of training task
- tensorflow_version: string, tensorflow version
- test_split: float32, test split proportion
- test_split_method: string, test split method
- num_free_parameters: uint64, exact number of trainable
  parameters
- widths: [uint32], list of layer widths used
- network_structure: string, marshaled json representation
  of network structure used
- platform: string, platform repetition executed on
- git_hash: string, git hash of version of experimental

Statistics about Per-Repetition Points of Minimum
KL-Divergence:

- Distribution of the Epoch of Minimum Test KL-
  Divergence of each Repetition:
  
  - test_kullback_leibler_divergence_min_epoch_min:
    [float32], earliest epoch at which test KL-
    divergence was minimized among all repetitions of the
    experiment
  - test_kullback_leibler_divergence_min_epoch_q1:
    [float32], first quartile of the epoch at which test
    KL-divergence was minimized
  - test_kullback_leibler_divergence_min_epoch_median:
    [float32], median epoch at which test KL-
    divergence was minimized
  - test_kullback_leibler_divergence_min_epoch_max:
    [float32], latest epoch at which test KL-
    divergence was minimized among all repetitions of the
    experiment
  - test_kullback_leibler_divergence_min_epoch_avg:
    [float32], average epoch at which test KL-
    divergence was minimized over all repetitions of the
    experiment

- Distribution of the Value of the Minimum KL-
  Divergence of each Repetition:
  
  - test_kullback_leibler_divergence_min_value_min:
    [float32], lowest minimum test KL-divergence
  - test_kullback_leibler_divergence_min_value_q1:
    [float32], first quartile minimum test KL-divergence
  - test_kullback_leibler_divergence_min_value_median:
    [float32], median minimum test KL-divergence
  - test_kullback_leibler_divergence_min_value_q3:
    [float32], third quartile minimum test KL-divergence
  - test_kullback_leibler_divergence_min_value_max:
    [float32], highest minimum test KL-divergence
  - test_kullback_leibler_divergence_min_value_avg:
    [float32], average minimum test KL-divergence

**d) Repetition Schema:** Each row of the repetition dataset
represents a single training run. Each repetition was instrumented to record various statistics at the end of each training
epoch, and those statistics are stored as epoch-indexed arrays
in each row. Runs are labeled with an 'experiment_id' which
was used to aggregate repetitions of the same experimental
parameters together in the summary dataset. An experiment_id
in the experiment dataset correspond to the same experiment_id
in the summary dataset. The columns of the full dataset are:
framework used
• hostname: string, hostname of system that executed this repetition
• seed: int64, random seed used to initialize this repetition
• start_time: int64, unix timestamp of start time of this repetition
• update_time: int64, unix timestamp of completion time of this repetition
• command: string, complete marshaled json representation of this repetition’s settings
• network_structure: string, marshaled json representation of this repetition’s network configuration
• widths: uint32, list of layer widths for the network used
• num_free_parameters: uint64, exact number of free parameters in the tested network
• val_loss: [float32], test loss at the end of each epoch
• val_mean_squared_error: [float32], test MSE at the end of each epoch
• val_mean_absolute_error: [float32], test MAE at the end of each epoch
• val_accuracy: [float32], test accuracy at the end of each epoch
• val_mean_squared_logarithmic_error: [float32], test MSLE at the end of each epoch
• val_root_mean_squared_error: [float32], test RMS error at the end of each epoch
• root_mean_squared_error: [float32], training RMS error at the end of each epoch
• mean_squared_error: [float32], training MSE at the end of each epoch
• mean_squared_logarithmic_error: [float32], training MSLE at the end of each epoch
• mean_absolute_error: [float32], training MAE at the end of each epoch
• hinge: [float32], training hinge loss at the end of each epoch
• hinge: [float32], test hinge loss at the end of each epoch
• squared_hinge: [float32], training squared hinge loss at the end of each epoch
• squared_hinge: [float32], test squared hinge loss at the end of each epoch
• cosine_similarity: [float32], training cosine similarity at the end of each epoch
• kullback_leibler_divergence: [float32], test KL-divergence at the end of each epoch
• kullback_leibler_divergence: [float32], training KL-divergence at the end of each epoch

Is any information missing from individual instances? No. In some cases a parameter value is NULL; this does not indicate missing data, but that the value is not applicable to that repetition or experiment. For example, runs with no regularization report NULL for $L^1$ regularization penalty because not only was the penalty 0, no regularization was applied what-so-ever. This allows us to distinguish between the case of activating the regularization code with zero penalty (the recorded penalty would be 0) and not activating the regularization code (the recorded penalty would be NULL).

Are relationships between individual instances made explicit? Yes. The repetition records corresponding to a summary record share the same experiment_id.

Are there recommended data splits? No. However, we organized the dataset by sweep (see Table II) through the parameter space. Salient dimensions, such as learning rate, shape, size, depth, label noise, and regularization level can be used to split the data in useful ways. Per-sweep slices of the summary dataset are available for download.

Are there any errors, sources of noise, or redundancies in the dataset? The dataset was collected over a period of time and multiple systems. Each system had different hardware and software configurations and the environment might have changed between runs. The experimental framework itself was also changed during the collection of these runs to enable greater flexibility and compatibility. Run records include many execution environment parameters including software version numbers, the git hash of the framework version, the hostname, and the operating system used. If differing operating environments introduced noise into the data, this metadata should help in detecting and filtering such noise or errors from the dataset. However, we have not detected any such variations. Experimental repetitions were repeated multiple times with varying random seeds; these matching repetitions will have the same experiment_id values. The number of repetitions of a given experiment is also present in its experiment summary record. This number can vary between experiments.

Is the dataset self-contained, or does it link to or otherwise rely on external resources? The dataset does not rely on external resources, but the training tasks utilized datasets from the Penn Machine Learning Benchmarks collection of datasets.

Does the dataset contain data that might be considered confidential? No.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? No.

3) Collection process: How was the data associated with each instance acquired? The worker process which executed the experiment recorded the data for that repetition in the database using the same unique id associated with the repetition task stored in the task queue. That same unique id is also stored in each run record of the dataset.

What mechanisms or procedures were used to collect the data? Our experimental framework was used to collect the data using the systems listed in Table V.

If the dataset is a sample from a larger set, what was the sampling strategy? The data was not sampled from a larger
Who was involved in the data collection process and how were they compensated? The authors, as paid employees of the Alliance for Sustainable Energy, LLC collected the data.

Over what timeframe was the data collected? The data was collected from 2021 to 2022. Collection start and end dates for each run are recorded in the dataset.

Were any ethical review processes conducted? No.

4) Preprocessing/cleaning/labeling: Was any preprocessing/cleaning/labeling of the data done? Runs and summary records were annotated with sweep set membership flags. No preprocessing or cleaning was done to the run table.

Was the “raw” data saved in addition to the preprocessed/cleaned/labeled data? The full dataset is composed primarily of the raw data which is unmodified and unfiltered.

Is the software that was used to preprocess/clean/label the data available? No. Sweep labeling was accomplished using simple database queries which marked experiments based on set membership of the parameters of each experiment in each sweep. Sweep labels are derived directly from the raw data and are provided only for convenience; they are not a fundamental component of the dataset.

5) Uses: Has the dataset been used for any tasks already? No.

Is there a repository that links to any or all papers or systems that use the dataset? No. However, additional resources can be found on the Framework’s Open Energy Data Initiative (OEDI) Record [48] and the Framework’s DOE Code record [49].

What (other) tasks could the dataset be used for? Informing hyperparameter initialization and search is one possibility.

Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses? No.

Are there tasks for which the dataset should not be used? The dataset should not be used in an unethical or unreasonable way. The dataset only tests a finite set of hyperparameter combinations and training tasks. Therefore, generalizing findings from this dataset beyond the bounds of the specific experiments recorded in it must be done with care and an acknowledgment that such extrapolations lie beneath a veil of speculative uncertainty.

6) Distribution: Will the dataset be distributed to third parties outside of the entity on behalf of which the dataset was created? Yes. It is publicly available.

How will the dataset be distributed? The dataset is posted as part of the Open Energy Data Initiative (OEDI) dataset repository: data.openei.org/submissions/5708 [48].

When will the dataset be distributed? It is currently available.

Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)? The dataset is distributed under the Creative Commons Attribution-ShareAlike 4.0 International Public License 11.

Have any third parties imposed IP-based or other restrictions on the data associated with the instances? No.

Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? No.

7) Maintenance: Who will be supporting/hosting/maintaining the dataset? Dr. Charles Tripp will maintain the dataset. It will be hosted by the United States Department of Energy (DOE) Open Energy Data Initiative (OEDI).

How can the owner/curator/manager of the dataset be contacted? Dr. Charles Tripp can be contacted at charles.tripp@nrel.gov.

Is there an erratum? Eratta and details are available here: github.com/openEDI/documentation/blob/main/BUTTER.md.

Will the dataset be updated? Updates will be posted in the dataset readme.

Will older versions of the dataset continue to be supported/hosted/maintained? If updated, older versions will remain accessible but may not be maintained. The versioning system and how to access previous versions will be described in the readme when new versions are released.

If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so? Please contact Charles Tripp, charles.tripp@nrel.gov, if you would like to augment the dataset. Any additions will be replicable, verified for consistency, and include instructions and software for doing so.

11Dataset License: creativecommons.org/licenses/by-sa/4.0/