DeepEutaxy: Diversity in Weight Search Direction for Fixing Deep Learning Model Training through Batch Prioritization

Hao Zhang and W.K. Chan, *Member, IEEE*

***Abstract*—Developing a deep learning (DL) based software system is slow. One of the critical issues is to conduct many trials and errors in developing a DL model that usually serves as the major component of such a system. A major reason for this inefficiency is the progress of gradual reduction of the gap between the DL model under training and the ground truths. Prior techniques commonly focus on optimizing such errors after the errors have formed. They are insensitive to how a training dataset is provided to the DL model under training in batches, making their approaches non-proactive to deal with such errors. In this paper, we propose DeepEutaxy, the first work to repair the model convergence problem from the batch prioritization perspective. Our key insight is that increasing the diversity (i.e., dissimilarity) of corresponding weights of complex DL models before and after each training step can make the models learn faster and optimize the training errors quicker. DeepEutaxy first trains a DL model with several epochs for initialization. It then partitions and continually prioritizes the training batches for subsequent training epochs based on our novel notion of diversity between the pair of models before and after training on each batch, capturing the strength of the search direction to deal with training errors impacted by that batch. The experiment on six deep learning models over the MNIST and CIFAR-10 datasets shows that DeepEutaxy can accelerate the convergence of DL models on these two datasets with speedups of 1.75 to 8.45 and 2.67 to 15.15 times with respect to the training and test accuracies, respectively. DeepEutaxy can also be integrated into existing techniques and compare favorably with the prior art in the experiment.**

***Index Terms*—Debug, fixing, deep learning models, efficiency, error reduction, model convergence, accuracy improvement**

I. Introduction

Deep learning (DL) [21] systems have shown their promises and have been widely applied in many areas such as image recognition [12], 3D modeling [2], self-driving cars [15], natural language processing [23], and machine translation [34]. A DL model usually consists of two parts: the model architecture (e.g., layers, neurons, and connections between neurons) and the corresponding parameters (e.g., weights and bias). Typically, the first layer and the last layer are called the *input layer* and the *output layer*, and the remaining layers are called *hidden layers*. Nodes contained in a layer are called *neurons*. Typically, neurons between layers are connected, and each connection is assigned with a specified value called *weight*. Input data for training and evaluating DL models are called *training dataset* (or *training samples*) and *test dataset* (or *test samples*).

Training a DL model refers to the process of updating weights of a DL model through iterative algorithms such as Stochastic Gradient Descent (SGD). Typically, in each iteration of the training process, each training sample is fed to the model once, and such an iteration is called one *epoch*. To reduce the computational cost and overcome the limitation of GPU memory during the training process, developers usually split the training data into a set of disjoint subsets, and each subset contains a certain number of samples. Such a subset is called a *batch*. One update of weights by one batch is called *one iteration* or *one step*. Given an input, the output of the DL model is the inference (or prediction) provided by the model.

In a software system development project that produces a DL model, developers usually write or revise a training script iteratively, collect or enlarge datasets, and train the DL model codified in the training script over such datasets. It is well-known that using different sets of training hyperparameters in different training phases will lead to different behaviors of the resultant DL models. Developers thus often try hard by *trials and errors* to improve the performance on those unseen inputs.

To do so, developers may alter the model structure of a DL model in the training script, enlarge the dataset, or testify whether the resultant DL model of the revised training phase will perform better (e.g., achieve a higher test accuracy). Usually, for the same training script, developers will apply many different sets of training hyperparameters, where each such set of hyperparameters initializes the training phase to produce a unique resultant DL model. On some other occasions, developers may alter the structure of the DL model codified in the training script. A change in the model structure will lead the whole process of hyperparameter selection and model training on the DL model with a revised structure to freshly re-start. The whole development process is often highly inefficient because the performance of a DL model could not be predicted in advance before the training phase of the DL model initialized with a specific set of hyperparameters.

In practice, many training scripts do not contain many lines of code, and their code mostly follows code templates available publicly. Apart from the dataset collection, another time-

Manuscript submitted July 26, 2020; date of current version April 27, 2021.

This work was supported in part by the GRF of Hong Kong Research Grants Council (project nos. 11214116 and 11200015), the HKSAR ITF (project no. ITS/378/18), and the CityU SRG (project nos. 7004882, 7005216, and 7005122) and Matching Grant (project no. 9678180).

Hao Zhang and W. K. Chan are with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: hzhang339-c@my.cityu.edu.hk and wkchan@cityu.edu.hk). W.K. Chan is the correspondence author.

consuming component of the development process is in the training phase. In this paper, we focus on fixing a kind of problem (see below) in this phase of the development process.

As we have stated, training a DL model usually requires iterating over the entire training dataset many times to produce a resultant model over the dataset. The underlying assumption is that a DL model under training only gradually learns the features from the samples over these iterations. Therefore, after many training epochs, the DL model under training will likely converge its behavior in recognizing features represented by the training samples, or the training process is stopped by developers. Afterward, among the series of DL models produced in a training process, developers often pick the model with the highest performance (e.g., attaining the highest test accuracy among all the models in the series) as the resultant DL model of the whole training process.

If a training phase can ***reduce errors faster and to a greater extent***, then the model is likely to ***converge faster***. Errors in the training process refer to the differences between the classification results of the model and the ground truth with respect to a given dataset. Having a faster rate of error reduction in model behavior for each initialized set of hyperparameters will help developers to speed up the entire development process of the corresponding DL system. For instance, a direct consequence of completing a training phase faster is that developers can testify more combinations of model structure and sets of hyperparameters under the same computational resources budget (compared to a slower training phase). The efforts saved can be allocated to new rounds of training to explore more possibilities of producing a DL model with higher performances, which will help manage the cost of development and shorten the maintenance phase of the DL-based development project.

Previous research has studied the effects of input data prioritization on reducing errors of DL models, such as using Curriculum Learning (CL) [3] and Self-paced Learning (SPL) [20]. Their common idea is to train a DL model with easier samples first and then feed it with more general and complex samples. In CL, a rank function is usually required to determine the difficulty of training samples. In SPL, a rank function is determined by the current status of the learning model, which is dynamically adjusted within the training process. They both ignore the internal behaviors (e.g., which set of neurons has been activated in classifying a particular input) of a resultant DL model. At the same time, in the research area of testing DL models (e.g., concolic testing [43], coverage testing [32], combinatorial testing [26], and mutation testing [28]), techniques often measure, use, or modify the internal behaviors and interactions between model structures and input data. Previous DL model debugging techniques like MODE [29], Apricot [49], and TRADER [44] take the internal structure and model similarities and differences into accounts.

We thus ask a couple of questions: *Is it feasible to utilize the internal behavior of a DL model in fixing the ineffective and inefficient error reduction problem incurred by the training process of that DL model? To what extent can such fixing be observed?*

For ease of our reference, we call the error reduction of a training process ***inefficient*** if it can be *repaired* to use fewer epochs to achieve the same error level. We also call the error reduction of a training process ***ineffective*** if the process can be *repaired* to achieve higher test performance (e.g., higher test accuracy). We generally refer to these two issues as the ***inefficient convergence*** problem of a training process.

In this paper, we present **DeepEutaxy**, a novel approach to fixing (alleviating) the inefficient convergence problem. It explores the connection between the search direction on setting weights to handle individual batches in the model optimization process and the batch prioritization in the overall training process. Given a DL model *M* with corresponding weights Θ and two training samples x_1 and x_2. Assume that Θ_1 and Θ_2 are weights that *M* are trained on x_1 and x_2 separately. If x_1 and x_2 are similar, i.e., belong to the same category, then, intuitively, only minor differences would be observed on their corresponding weights matrices, i.e., Θ_1 resembles Θ_2. Training a DL model on x_1 could weaken the effects of further training on x_2 because some common features have been learned from x_1 already. Our key insight is that maximizing the diversity (i.e., dissimilarity) of weights before and after a training step can help to accelerate the training effects and exploit the full potential of training samples, alleviating the inefficient convergence problem in training a DL model.

To the best of our knowledge, DeepEutaxy is the first work building on top of the above insight. Its general idea is as follows. Suppose that there is a DL model M_0 with its training dataset T_0. DeepEutaxy first trains M_0 with a small number of epochs to obtain a partially-trained DL model. Then, the training dataset T_0 is partitioned into a set of subsets $D = \{d_1, d_2, \dots, d_n\}$, each of which is called batches. DeepEutaxy measures the diversity between the weights of the DL model and the gradients of the model with respect to each of these batches. It then trains this partially-trained model with these batches in descending order of their degree of diversity over a certain number of epochs. After that, it repeats the above process. We have evaluated DeepEutaxy using the MNIST and CIFAR-10 datasets on six DL models distinguished by their model complexity. The experiment on six deep learning models over the MNIST and CIFAR-10 datasets shows that DeepEutaxy can accelerate the convergence of DL models on the MNIST and CIFAR-10 datasets with speedups of 1.75 to 8.45 times and 2.67 to 15.15 times in terms of training accuracy and test accuracy, respectively. DeepEutaxy also compared favorably with the prior art in the experiment.

The main contribution of this paper is threefold:

- This paper presents DeepEutaxy, the *first* work to show the feasibility of neural network training acceleration by batch prioritization. DeepEutaxy innovatively explores the impact of the differential behaviors of the DL model exhibiting over individual batches before and after training on these batches. It captures the search direction to handle these batches and explores to prioritize batches with stronger search direction indicators over the weaker ones.

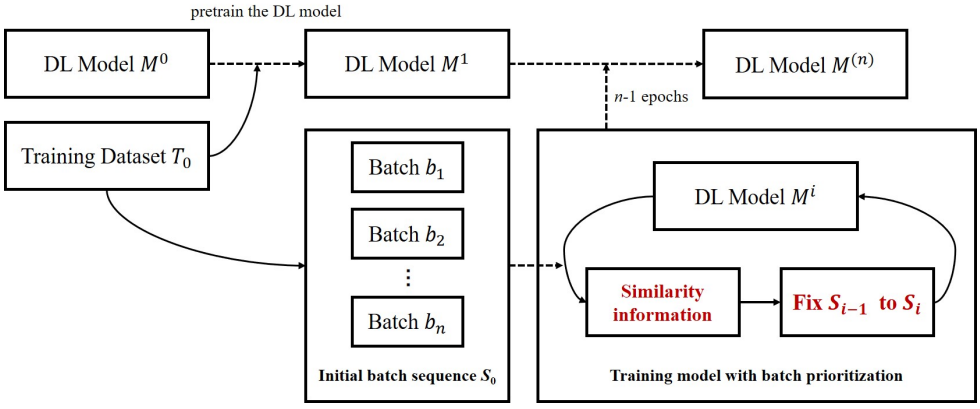

Fig. 1 Overall process of DeepEutaxy

- This paper presents an evaluation on DeepEutaxy. The results demonstrate the potential of differential batch prioritization in alleviating the inefficient convergence problem (for error reduction) in the training process of a DL model.
- This paper presents the *first* empirical study on the effect of batch reordering in neural network training. It shows that batch prioritization is a significant factor in affecting model convergence positively.

The organization of the remaining sections is as follows. Section II revisits the preliminaries. Section III presents DeepEutaxy, followed by its evaluation in Section IV. Section V discussed related works. Section VI concludes this work.

II. Preliminaries

A. Deep Learning Models

Typically, a deep learning model consists of two parts: model architecture (e.g., neurons, layers, and the connections between layers) and corresponding parameters (e.g., weights). Usually, the architecture of a DL model is manually designed. The weights of a DL model are firstly initialized with random values and then updated through iterative learning algorithms. The update process is realized by *forward propagation* and *backward propagation* [35]. Given an input or a set of inputs, the gradients of errors between the inference of the model and the ground truth are calculated at the weight level.

Training a DL model is a time-consuming task. DL models such as VGG16 [39], ResNet [12], and AlexNet [17] usually contain millions of parameters. Tens to hundreds of epochs are needed for updating and tuning parameters. A model is of ***low model complexity*** if it contains very few hidden layers (e.g., 5-10). These DNN models (e.g., LeNet-5 [22]) are proposed when Deep Learning flourished in the early 2010s. Modern DL models (e.g., ResNet [12]) usually contain tens of layers and have solved the gradient vanishing problem in model optimization. We refer to them as models with ***high model complexity***.

B. Gradient Descent and Stochastic Gradient Descent

Gradient Descent (GD) is an optimization algorithm used to search for a global optimum of a function. In the area of DL models, GD is to update parameters iteratively. Formally, the training process of a DL model is to optimize the following:

$$\mathbb{E}_{(x,y)\sim D}[\max(L(x,y,\theta))] \quad (1)$$

where D is the training dataset, $L(\cdot)$ is the loss function, $\mathbb{E}$ is the expectation, θ is the parameters (e.g., weights). In GD, θ is updated through the following equation:

$$\theta^* = \theta + \eta\nabla_\theta L(x,y,\theta) \quad (2)$$

where θ and θ^* are the parameters before and after the update, η is the learning rate, $\nabla_\theta L(x,y,\theta)$ is the ***gradient*** of the loss function with respect to the model parameter θ, x and y are the given input and the corresponding ground truth, respectively.

A training process can be viewed as an iterative procedure of finding the global optimum of θ, where $\nabla_\theta L(x,y,\theta)$ and η determine the *search direction* and the *step size* in the weights space. In each iteration of GD, all training samples are involved in calculating the overall gradients, which provide a relatively accurate search direction to the global optimum.

Stochastic Gradient Descent (SGD), unlike GD, uses a subset of training samples to calculate the gradients and update parameters. Compared to GD, SGD converges much faster because less data involves in each iteration. However, the loss function may not be as minimized as in the case of GD. SGD is more popularly applied in practice because a typical training dataset contains many samples, making the processing of all samples in each iteration less attractive.

C. Curriculum Learning and Self-Paced Learning

Inspired by the learning process of human beings, the basic idea of curriculum learning (CL) [3] and self-paced learning (SPL) [20] is to train a DL model starting from "easy" samples first and gradually increase the difficulty of the learning task. Instead of training a DL model over the whole training dataset, CL evaluates the difficulty of each sample through a manually crafted function $\Psi(\cdot)$ and trains the model with easier samples first. In practice, developers need to determine the rank function themselves, which is subjective. Besides, such rank functions are likely to differ by different learning tasks and model architectures to ensure their effectiveness, which requires domain knowledge from experts.

Unlike CL, SPL selects samples for training adaptively and objectively. Formally, the loss function of SPL is as follows:

Algorithm 1 DeepEutaxy

Input: *train_x, train_y* ← inputs and labels of training dataset; *num_per_batch* ← number of samples contained in one batch; *epochs* ← number of iterations for training the model; *iter_count* ← number of iterations before sorting the batch sequence; *model* ← the machine learning model to be trained

Output: trained machine learning model

```
batch_seq ← SplitDataset(train_x, train_y, num_per_batch)
for each batch in batch_seq do
    TrainOnBatch(model, batch)
end for
for epoch in epochs do
    if epoch mod iter_count equals 0 do
        similarity_dict ← GetSim(model, batch_seq)
        batch_seq ← SortSim(batch_seq, similarity_dict)
    end if
    for each batch in batch_seq do
        TrainOnBatch(model, batch)
    end for
end for
return model
```

$$L_{SPL} = \sum_{i=1}^{n} v_i L(x_i, y_i, \theta) - \lambda \sum_{i=1}^{n} v_i \tag{3}$$

where $L(\cdot)$ is the loss function, x_i and y_i are the given input and the corresponding ground-truth label. v_i is a binary indicator that takes the value 0 or 1. λ is a threshold. If the value of $L(x_i, y_i, \theta)$ is higher than λ, then v_i is set to 1 and $L_i = v_i(L(x_i, y_i, \theta) - \lambda)$ is added to L_{SPL}. Otherwise v_i is set to 0.

To make L_{SPL} more practical, λ is usually adjusted dynamically during the training process. That is, λ is updated by the following equation:

$$\lambda^{(i)} = \alpha\lambda^{(i-1)} \tag{4}$$

where $\lambda^{(i)}$ is the value of λ after the i-th epoch, and α is a growing factor.

From the loss function of SPL, we can observe that those samples with losses smaller than a given threshold will be involved in the current training epoch. Informal speaking, the values of losses could be regarded as the indicator telling if samples are easy or not to learn by DL models.

III. DeepEutaxy

A. Overview

DeepEutaxy aims at reducing errors of a DL model in training. It is a ***dynamic fixing strategy***. It assumes that the training script of a DL model is available. The basic idea of DeepEutaxy is to monitor the search direction of the DL model in handling individual batches (realized as diversity in weights of the DL model before and after training with each batch) followed by reordering the batch sequence for training the DL model over that reordered batch sequence. In this way, in each epoch, the DL model under training is learned to be aware of "diverged" batches while adjusting the other batches.

The overall process of DeepEutaxy is depicted in Fig. 1. DeepEutaxy firstly trains a DL model over the training samples with several epochs for initialization. It then partitions the training dataset into small batches. It measures the diversity (i.e., dissimilarity) between the weight matrices of this initialized DL model before and after applying that batch. Then, it trains the initialized DL model by batches one by one in descending order of the measured diversity to obtain the resultant DL model.

Unlike existing training acceleration approaches [3][13][20][41], DeepEutaxy collects internal information of a DL model during training, extracting insights on the evolution of the internal characteristics of the DL model in training. In particular, our insight is that at the early stage of training of a complex DL model, increasing the diversity of weights in each step can effectively guide the model to move to a search direction toward the global optimum, which also align with the notion of gradient descent as an efficient and effective strategy to train a DL model.

Recall that in each step, gradients derived from the loss function in a training phase provide a search direction for updating weights. To ensure the generalization of DL models, training models with samples from different categories are usually required. Through DeepEutaxy, we formulate a novel mechanism toward maximizing the diversity of internal behaviors by comparing diversity (or dissimilarity) between weight matrix Θ before feeding a given batch b and the gradient G of the model with respect to b. The calculation of this novel notion of diversity is formulated as follows:

$$Sim(\Theta, G) = \sum_{l} CosSim(\theta_l, g_l) \tag{5}$$

$$CosSim(\theta_l, g_l) = \frac{vec(\theta_l) \times vec(g_l)}{\|vec(\theta_l)\|_2 \|vec(g_l)\|_2} \tag{6}$$

where $\Theta = \{\theta_1, \theta_2, \dots, \theta_l\}$ is a list of weights matrices, θ_i is the weights matrix for the i-th layer, and l is the total number of layers. $G = \{g_1, g_2, \dots, g_l\}$ is a list of gradients. Note that G is

a matrix that has the same shape as the weight matrix. $CosSim(\cdot)$ is the cosine similarity for measuring the similarity of two vectors. $\|\cdot\|_2$ is the L_2 norm. $vec(\cdot)$ flattens a matrix into a vector in row-major order. For example, given a matrix $\theta = \begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$, $vec(\theta) = [1,2,3,4]$.

In the above design, we use the cosine similarity to quantify the degree of dissimilarity between the weight matrix and its gradient. Our intuition is that two vectors with the same search direction could be identified as equal by cosine similarity. Our insight is that in a DL model with a weight matrix Θ, suppose there are two given samples (referred to as x_1 and x_2) and their derived gradients are G_1 and G_2, then the condition $Sim(\Theta, G_1) < Sim(\Theta, G_2)$ will provide a clue that x_1 is likely to provide a more significant change in weights in terms of search direction than x_2 in updating the weight matrix Θ. Furthermore, if $Sim(\Theta, G_2)$ is close to 1, G_2 is likely to update weights in the current direction to train the DL model, i.e., enhancing the features that have been kept in the weight matrix. It indicates that such an update will have a diminishing effect on adding "new" information to Θ, which may aggravate the overfitting problem.

Suppose that Θ is the weight matrix of a DL model, and $G_{b_1}, G_{b_2}, \dots, G_{b_n}$ are the gradients of batches $b_1, b_2, \dots, b_n$, respectively. We can thus compute the diversity for each batch, denoted as $Sim(\Theta, G_{b_i}), i = 1,2, \dots, n$. Batches are prioritized and fed to the DL model based on the measured degree of diversity in descending order (with tie cases resolved randomly).

B. Batch Prioritization by Epoch

The detailed algorithm of DeepEutaxy is shown in Algorithm 1. In Algorithm 1, the training dataset and some parameters are given as inputs to the algorithm. The parameter *num_per_batch* is the total number of samples contained in one batch. *Epochs* is an input parameter to control the total number of iterations. *iter_count* is used to control the number of iterations before re-prioritizing the batch sequence.

In line 1, the training dataset is divided into a set of batches, and those batches are randomly sorted in a sequence called a batch sequence. Every batch is fed into the model once for initializing the model (lines 2-4). Then, DeepEutaxy feeds the initialized model with a batch sequence over some number of epochs (lines 5-14). After that, the similarity information calculated by Equations (5) and (6) for each batch is collected. The batch sequence is reordered based on dissimilarity in descending order (line 7 and line 8), i.e., batches with larger diversity are fed to the model with higher priority. After batch prioritization, the model is fed with a newly prioritized batch sequence (lines 10-12).

To clarify our algorithm further, we make the following notes: Algorithm 1 uses *iter_count* to iterate training steps several times (rather than after each step) before re-prioritizing the batch sequence. Each time DeepEutaxy feeds batches to the model, the training accuracy and test accuracy of the corresponding DL model are evaluated for further analysis. Moreover, in Algorithm 1, DeepEutaxy continues the training process even if the convergence criterion is satisfied, which can be easily modified to terminate it when convergence (e.g., having more or less the same level of test accuracy) is reached. For brevity, we do not show this variant of Algorithm 1 in this work.

IV. Experiment

A. Experimental Setup

We implemented our experiment on top of Keras 2.3.1 [16] and TensorFlow 2.2.0 [45], which were popular deep learning libraries in Python. All experiments were performed on Ubuntu 18.04 running on Intel(R) Xeon(R) Gold 6136 CPU@3.00 GHz with 12 cores and four NVIDIA GeForce 2080-Ti GPUs with the VRAM size of 11×4 GB. We followed the general setting in prior studies [10][49] and repeated our experiment five times to ensure consistency. The experimental results and source code are available at https://github.com/DemoAuguste/DeepEutaxy.

Datasets. The experiments and evaluations were conducted using the MNIST [22] and CIFAR-10 [17] datasets that were widely used for image classification training and testing researches as well as testing and debugging researches on DL models [10][12][37][49]. The MNIST dataset contained 60,000 training samples and 10,000 test samples. The CIFAR-10 dataset contained 50,000 training samples and 10,000 test samples.

DL models. We used five DL models for evaluation: a CNN model (denoted as CNN_1), LeNet-5 [22], ResNet-20 and ResNet-32 [12], and MobileNetV2 [37]. We chose these models because the first two represented models with low model complexity, and the last three represented models that were high in model complexity. In this way, we can study the general trend across DL models with different model complexity and contrast their differences.

The structures of CNN_1 and LeNet-5 are summarized in TABLE I. Their structures were typical and representative. CNN_1 contained two convolutional layers with 32 and 64 kernels followed by a max-pooling layer. Two dropout layers and a fully connected layer were added to prevent the model from overfitting. The output part consisted of one fully connected layer followed by a *softmax* layer. LeNet-5 [22] was a representative DL model that had shown promising performance in many scenarios. The activation function used in all DL models was ReLU, which was commonly applied in neural networks research.

TABLE I Structure of CNN_1 and LeNet-5

Model	CNN_1	LeNet-5
Block 1	Conv(32) Conv(64) MaxPooling	Conv(6) MaxPooling
Block 2	Dropout(0.25) Flatten Dense(128) Dropout(0.5)	Conv(16) MaxPooling
Block 3	-	Dense(120) Dense(84)
Output	Dense(10) Softmax	Dense(10) Softmax

TABLE II SETTINGS FOR TRAINING DL MDOELS

Model	No. of parameters	Optimizer	learning rate
CNN_1	0.87M	Adagrad	0.1
LeNet-5	0.44M	Adagrad	0.1
ResNet-20	0.27M	Adadelta	-
ResNet-32	0.47M	Adadelta	-
MobileNetV2	2.29M	SGD	0.1
MobileNetV2-2		CLR	[0.0001, 0.1]

TABLE III THE PERFORMANCE OF BL

Model	Dataset	No. of epochs	Training accuracy	Test accuracy
CNN_1	MNIST	200	98.87%	98.52%
LeNet-5	MNIST	200	99.09%	97.93%
ResNet-20	CIFAR-10	200	99.76%	68.54%
ResNet-32	CIFAR-10	200	99.93%	75.21%
MobileNetV2	CIFAR-10	200	98.98%	51.97%
MobileNetV2-2			99.10%*	41.94%*

*Results of the model trained with CLR.

For ResNet-20, ResNet-32, and MobileNetV2, we used the existing implementations downloaded from [5] and [30].

Settings for training DL models are presented in TABLE II. Columns 2-4 show the number of parameters in the model, optimizers, and initial learning rate, respectively. Note that the training efforts could be sensitive to the learning rate. For example, as shown in [40], training DL models with different adjustment strategies of learning rate could have different convergence speeds. To evaluate the performance of DeepEutaxy with different learning rate adjustment strategies, we applied different optimizers to different DL models. We applied Adagrad [9], Adadelta [48], CLR [40], and SGD as optimizers for training CNN_1 and LeNet-5, ResNet-20 and ResNet-32, and MobileNetV2, respectively. Adagrad, Adadelta, and SGD have been embedded into Keras, and we applied existing CLR implementation [6]. We referred to MobileNetV2 applying CLR as MobileNetV2-2.

Formally, Adagrad updates weights by following equations:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\sum_{i=1}^{t} g_i^2 + \epsilon}} \cdot g_t \tag{7}$$

where $g_t = \nabla_\theta L(x, y, \theta)$ is the gradient calculated at the t-th step, ϵ is a smoothing term avoiding division by zero, η is predetermined the learning rate.

Adadelta extends Adagrad and updates weights as follows:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{8}$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\theta_t^2 \tag{9}$$

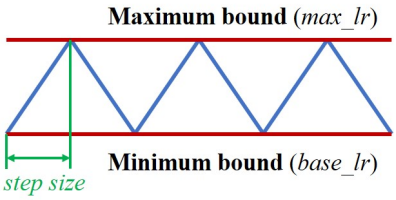

Fig. 2 Cyclical learning rates adjustment strategy. The red lines represent the maximum bound and minimum bound of learning rate. The blue lines represent that the learning rate changes between bounds. Step size is the number of iterations in half a cycle.

$$\Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_t + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} \tag{10}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{11}$$

where E is the expectation, γ is a predetermined parameter. Note that in Adadelta, there is no need to set a learning rate manually.

Cyclical Learning Rates (CLR) [40] is a dynamic strategy to adjust the learning rate cyclically between a given range during the training process. As shown in Fig. 2, CLR adjusts the learning rate between two predetermined bounds. We note that there is no general rule to set these bounds. Our experiment followed the setting in [7] and set the minimum bound and maximum bound to 0.0001 and 0.1, respectively. The step size is set to 200, arbitrarily.

Baseline (BL). In the experiment, the training accuracy of each DL model that was trained by the corresponding original training script (i.e., feeding batch sequences with a fixed order) was chosen as a baseline (BL) for evaluating the effectiveness of DeepEutaxy. Unlike DeepEutaxy, in BL, the order of batch sequence was determined and fixed after a random initialization. We note that this is a typical setting in training scripts.

Comparing DeepEutaxy with Self-Paced Learning (SPL). We also compared DeepEutaxy (DE) with Self-Paced Learning (SPL). We followed the existing implementation [38] where we set λ to 0.1 and α to 1.3 for Equations (2) and (3).

We did not compare DeepEutaxy with Curriculum Learning (CL) because CL required a predetermined rank function to decide whether a given input was easy or not. It could be subjective for us to design such a ranking function for each dataset used in our experiment.

Other settings and parameters. We set the batch size to 1,000. Moreover, we set *iter_count* in Algorithm 1 to 5 and 10 in training all six DL models on the MNIST and CIFAR-10 datasets, respectively.

Convergence criterion. A widely used method to observe model convergences is to monitor the training accuracy (or the change in gradients or losses) and check whether it reaches almost the same level. Following the common practice, we used the following criterion to determine whether a DL model converged:

$$\|\nabla L(x, x')\|_2 \leq \varepsilon \tag{12}$$

where $L(\cdot)$ is the loss function, ∇ is the gradient operator, x and x' are the predicted value and ground-truth label, $\|\cdot\|_2$ is the L_2 norm (Euclidian distance), ε is a predefined threshold. In our experiment, ε is set to 0.01. In each step, we evaluated the gradients obtained in that training step. If the L_2 norm of gradients was higher than ε, and we recorded the number of epochs that had been proceeded. If all 200 epochs had been exhausted and the DL model had not satisfied Equation (12), we deemed the model "not converged".

B. Results and Data Analysis

1) Results of DL models trained by BL

The results of DL models trained by the standard training

TABLE IV CONVERGENCE RESULTS OF DL MODELS ACHIEVED BY BASELINE (BL), SELF-PACED LEARNING (SPL), AND DEEPEUTAXY (DE)

Model Complexity	Model	BL		SPL		DE		Number of epochs for reaching the convergence criterion		
		Train acc.	Test acc.	Train acc.	Test acc.	Train acc.	Test acc.	BL	SPL	DE
Low	CNN_1	98.87%	98.52%	99.03%	98.86%	**99.83%**	**99.24%**	81.8	82.6	**55.8**
	LeNet-5	99.09%	97.93%	99.03%	98.36%	**99.88%**	**98.78%**	104.6	109	**64.6**
High	ResNet-20	99.76%	68.54%	90.17%	**71.48%**	**99.98%**	66.60%	82.6	not converged	**37.8**
	ResNet-32	99.93%	**75.21%**	90.82%	73.30%	**100.00%**	69.58%	106.6	not converged	**36.2**
	MobileNetV2	98.98%	51.97%	85.84%	**64.72%**	**99.98%**	58.86%	121.3	124.5	**64.6**
	MobileNetV2-2	99.10%	41.94%	86.45%	**62.41%**	**100.00%**	57.91%	155.8	not converged	**70.0**

TABLE V RESULTS OF MAXIMUM ACCURACIES OF DL MODELS (TRAINING ACCURACY)

Model	Max. training accuracy			Number of epochs to reach the highest training accuracy						
	BL	SPL	DE	BL (A)	SPL	SPL (same level as BL) (B)	Speedup (A/B)	DE	DE (same level as BL) (C)	Speedup (A/C)
CNN_1	99.10%	99.36%	99.99%	191.0	196.4	44.0	4.34	193.6	22.6	**8.45**
LeNet-5	99.31%	99.22%	100.00%	193.0	196.2	56.0	3.45	171.8	41.2	**4.68**
ResNet-20	99.95%	90.17%	100.00%	116.4	155.4	*failed*	-	38.2	36.2	**3.22**
ResNet-32	99.98%	90.82%	100.00%	171.8	166.0	*failed*	-	36.2	35.6	**4.83**
MobileNetV2	98.08%	86.80%	100.00%	149.2	197.4	*failed*	-	160.2	85.4	**1.75**
MobileNetV2-2	99.19%	86.45%	100.00%	194.2	199.0	*failed*	-	70.0	53.0	**3.66**

TABLE VI RESULTS OF MAXIMUM ACCURACIES OF DL MODELS (TEST ACCURACY)

Model	Max. test accuracy			Number of epochs to reach the highest test accuracy						
	BL	SPL	DE	BL (A)	SPL	SPL (same level as BL) (B)	Speedup (A/B)	DE	DE (same level as BL) (C)	Speedup (A/C)
CNN_1	98.67%	99.09%	99.35%	176.0	179.2	63.5	2.77	108.8	14.0	**12.57**
LeNet-5	98.04%	98.47%	98.86%	118.2	186.6	39.8	2.97	99.8	7.8	**15.15**
ResNet-20	70.33%	71.48%	69.02%	154.6	162.8	126.0	**1.22**	128.8	*failed*	-
ResNet-32	75.92%	73.30%	69.92%	163.4	166.0	*failed*	-	39.8	*failed*	-
MobileNetV2	52.05%	65.24%	59.03%	75.4	125.6	23.2	**3.25**	115.6	28.2	2.67
MobileNetV2-2	42.83%	62.41%	58.17%	36.0	191.2	14.2	2.54	71.8	12.6	**2.86**

scheme (BL) are shown in TABLE III. Column 1 shows the DL models used in our experiment. Columns 2 and 3 present the dataset used and the number of epochs for training each model. We set the number of epochs to 200. The training accuracy and test accuracy are shown in Columns 4 and 5, respectively. Note that the training accuracy and the test accuracy were the values at the epoch with the convergence criterion satisfied right after it or at the 200-th epoch if the model did not trigger the convergence criterion.

It is worth noticing that in our training process, ResNet-20 did not achieve the same accuracy as presented in [12]. This is because in [12], data augmentation was applied in its training process, which had significantly altered the obtained accuracy of these models by creating transformed versions of images, expanding the training dataset with new, plausible samples. We also note that in our experiment, only the original CIFAR-10 dataset was used to control the size of a training batch.

2) Results of DL models trained by SPL and DE

The experimental results are summarized in TABLE IV, TABLE V, and TABLE VI.

TABLE IV shows the number of epochs used before convergence. Note that we used Equation (12) to determine if the model converged in the experiment and trained each model with a maximum of 200 epochs. Columns 3-8 show the training and test accuracies of the DL model when triggering the convergence criteria by applying BL, SPL, and DE, respectively. Numbers in bold are the highest accuracies achieved across BL, SPL, and DE in the same row. Columns 9-11 show the number of epochs for training before triggering the convergence criterion.

DE required the least training epochs for convergence across all three methods using 36 to 70 epochs on average. SPL did not converge after being trained with 200 epochs in 3 out of 4 high-complexity models. For the models converged, BL and SPL spent 105.3% and 69.8% more epochs than DE to converge the model on average, respectively. Note that SPL requires some hyperparameters (e.g., λ and α in Equations (3) and (4), learning rate, optimizer, and training script), which seems to indicate that developers may need to pay more efforts to discover a correct set of SPL's hyperparameters to make their DL models converge.

TABLE V presents the results of reaching the maximum accuracies of corresponding DL models. Columns 2-4 show the maximum training accuracies achieved by BL, SPL, and DE, respectively. Columns 5, 6, and 9 show the number of epochs for reaching the corresponding maximum training accuracies of BL, SPL, and DE, respectively. Columns 7 and 10 show the number of epochs that SPL and DE reached the same or higher accuracy compared to the highest accuracy achieved by BL. Columns 8 and 11 are computed by the value in Column 5 divided by Column 7 and Column 10, respectively. Take CNN_1, for example. BL, SPL, and DE achieved the highest training

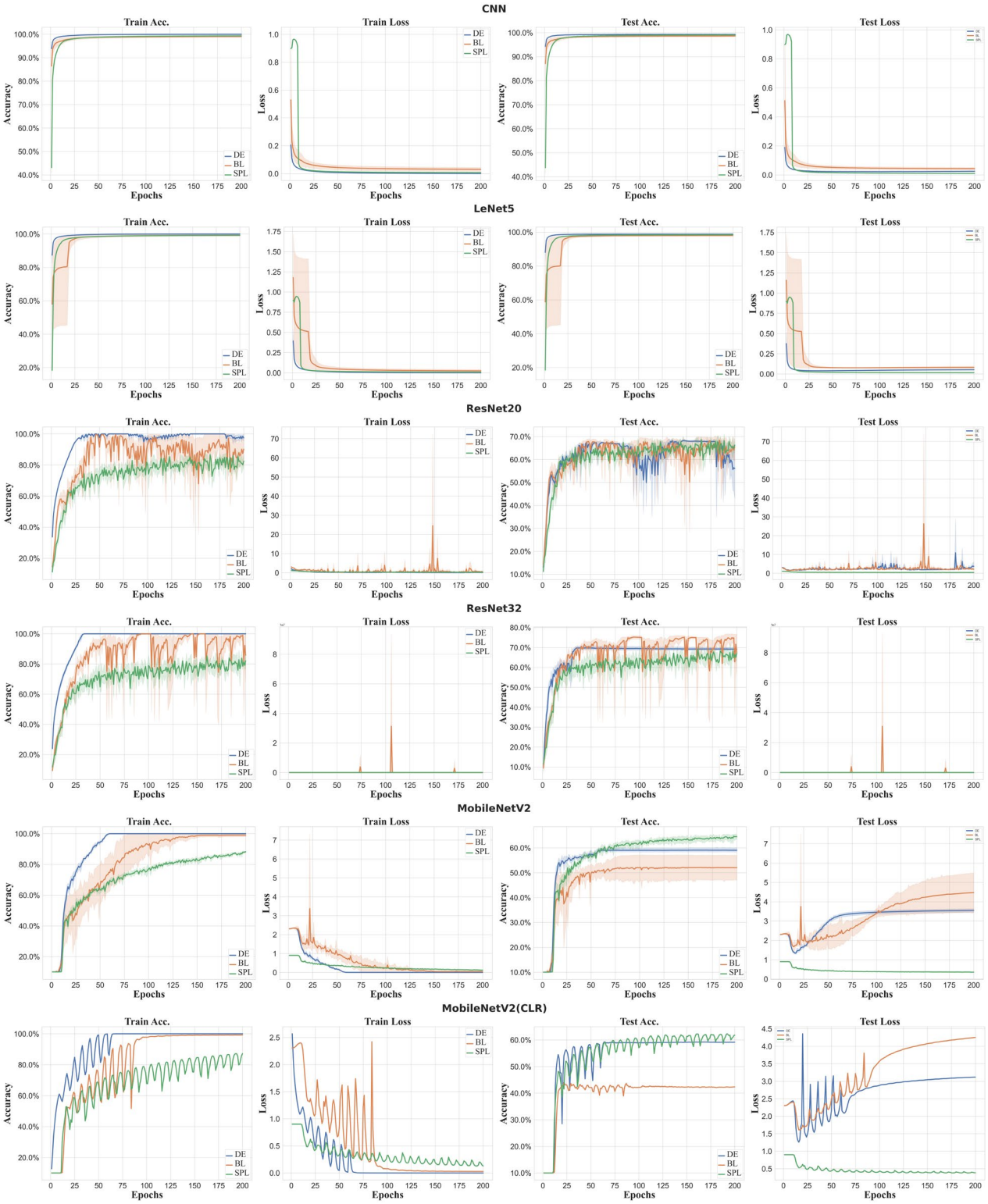

Fig. 3 The training and test accuracy curves of six DL models in the experiment. For each model, there are four subplots. For each subplot, there are three lines with error bands, representing training accuracy, training loss, test accuracy, test loss trained by the baseline (BL), self-paced learning (SPL), and DeepEutaxy (DE), respectively.

accuracy after training with 191, 44, and 23 epochs on average, respectively.

From TABLE V, DE reached or exceeded the same level of training accuracies compared to BL with a speedup ranging from 1.75 to 8.45 times across all six DL models. SPL achieved speedups of 4.34 and 3.45 times on CNN_1 and LeNet-5, respectively. On ResNet-20, ResNet-32, MobileNetV2, and MobileNetV2-2, SPL can only achieve lower accuracies than BL, which we marked in the corresponding entry in the table as *failed*.

TABLE VII RESULTS OF TRAINING MODELS BY SPL, DE, AND SPL+DE

Method	Model					
	ResNet-20			ResNet-32		
	Max train acc.	Max test acc.	Additional epochs for reaching the convergence criterion	Max train acc.	Max test acc.	Additional epochs for reaching the convergence criterion
Base*	90.17%	71.48%	-	90.82	73.30%	-
SPL	91.34%	71.64%	*failed*	90.65%	73.19%	*failed*
DE	**100%**	73.49%	**8**	**100%**	**75.19%**	**7**
SPL+DE	98.11%	**73.65%**	27	98.51%	74.16%	23

*the accuracy of the base model. We further train the base model with 50 epochs by SPL, DE, and SPL+DE, respectively.

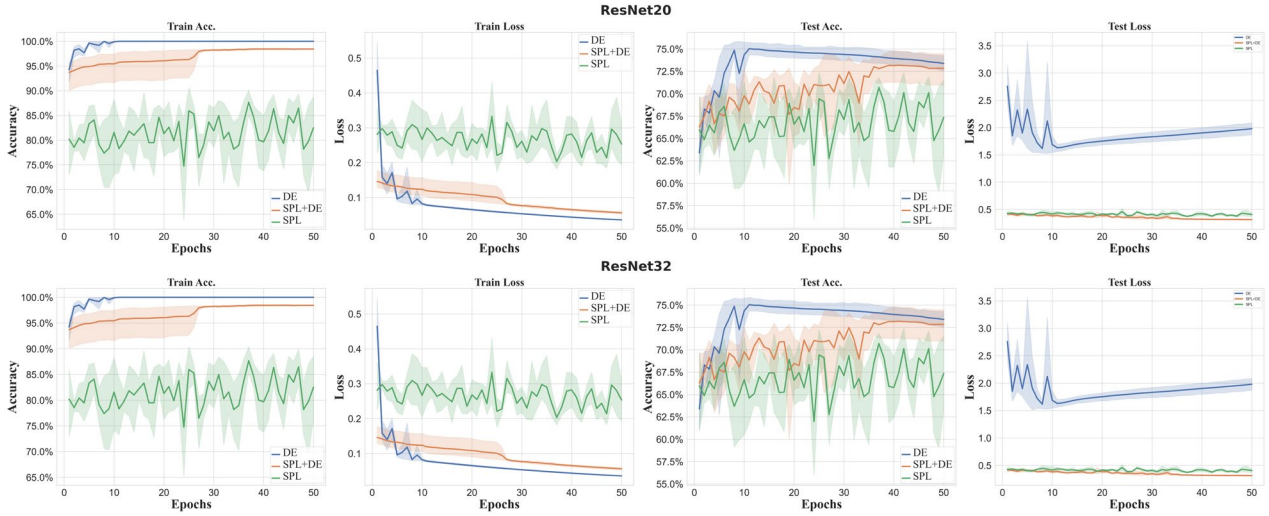

Fig. 4 The training and test accuracy curves of ResNet-20 and ResNet-32.

DE achieved the highest training accuracies across BL, SPL, and DE. Besides, for SPL, we found that for DL models with high complexity, i.e., ResNet-20, ResNet-32, MobileNetV2, and MobileNetV2-2, fps between training accuracies achieved by SPL and BL. It may indicate a higher hidden threat of the DL models trained with SPL.

TABLE VI shows the results of test accuracies and can be read similarly to TABLE V. The results in TABLE VI show that DE achieved speedups on CNN_1 and LeNet-5 by 12.57 and 15.15 times, respectively. For ResNet-20, SPL achieved speedup by 1.22 times, but DE did not achieve the same level of test accuracy. For ResNet-32, neither SPL nor DE reached the same level of test accuracy. For MobileNetV2 and MobileNetV2-2, DE and SPL achieved speedups by 2.67 and 2.86 times, and by 3.25 and 2.54 times, respectively. The performance of DE and SPL on MobileNetV2 and MobileNetV2-2 are similar.

DE achieved higher test accuracies than BL except for ResNet-20 and ResNet-32. Furthermore, SPL outperformed DE with respect to test accuracy on DL models with high complexity.

As shown in Fig. 3, we plot the accuracies and losses (measured by the corresponding loss functions used in the training scripts) of the training processes. In Fig. 3, the *x*-axis represents the number of epochs of a training process consumed, and the *y*-axis represents the accuracy (or loss) of the corresponding DL model after training for the number of epochs, as indicated by the *x*-axis. For each model, there are four subplots, presenting the training accuracy, training loss, test accuracy, and test loss, respectively. In each subplot, there are three lines with error bands. The blue line, orange line, and green line represent the training results of DE, BL, and SPL, respectively. From the plots, we can see that DE has a more significant effect on the training process of these DL models in the high model complexity category.

In these plots, the training accuracy and the test accuracy achieved by DE could reach the same or almost the same levels as BL if not higher.

For MobileNetV2 and MobileNetV2-2, the test losses increased after reaching the minimum. It indicates that the two models have been overfitted as measured by the test datasets. From the figure, DE appears to exhibit a better effect than BL to alleviate the magnitude of extreme test accuracies.

From the plots, it appears that SPL has much lower test losses compared to BL and DE. The loss function of SPL is different from BL and DE. Recall that BL and DE applied Cross-Entropy Loss, but SPL applied Equation (3) to calculate the loss. Due to the negative term in Equation (3), SPL could achieve smaller losses compared to BL and DE.

In particular, in these plots, there are significant gaps between DE and BL on MobileNetV2 and MobileNetV2-2. The difference in training error reduction between them is also significant. For SPL, it achieved higher test accuracies on MobileNetV2 and MobileNetV2-2. However, as shown in TABLE IV, SPL did not converge on ResNet-20, ResNet-32, and MobileNetV2-2.

3) The Effectiveness of DE on trained DL models

We further implemented DE on top of SPL. We noted that

DE and SPL are orthogonal in optimizing DL models, and our experiment presents the first piece of evidence to explore the combined effort of these two aspects.

In the training process of SPL, before feeding training batches into the model, the batches are prioritized by DE. We refer to this combined technique as SPL+DE. The main difference between DE and SPL+DE is that the loss function of SPL+DE is the same as SPL.

From TABLE IV, we noticed that ResNet-20 and ResNet-32 trained by SPL might not converge. Besides, from TABLE V and TABLE VI, DE failed to achieve comparable test accuracy to BL on ResNet-20 and ResNet-32. Note that those models had been trained with 200 epochs. To further evaluate the effectiveness of DE, we further trained the resultant ResNet-20 and ResNet-32 models (that have been trained by SPL with 200 epochs) with 50 more epochs by SPL, DE, and SPL+DE, respectively. We refer to the resultant model trained by SPL with 200 epochs as *Base*.

The results are shown in TABLE VII and Fig. 4. In TABLE VII, Columns 2-7 show the maximum training accuracy, maximum test accuracy, and the number of additional epochs for reaching the convergence criterion with respect to ResNet-20 and ResNet-32. Numbers in bold present the highest accuracy or the least number of epochs to make the model converged across SPL, DE, and SPL+DE. If a model cannot converge in the experiment, we marked it as *failed*.

Neither training accuracy nor test accuracy of ResNet-20 and ResNet-32 improved noticeably by applying SPL.

On the other hand, DE and SPL+DE achieved maximum training accuracy of 100% and 98.11% on ResNet-20, 100% and 98.51% on ResNet-32, respectively. Recall that the training accuracy of SPL was significantly lower than BL in TABLE V, and after applying DE on top of SPL, the training accuracies of SPL+DE were significantly improved.

The improvement in test accuracy achieved by DE and SPL+DE was also significant: DE and SPL+DE achieved increases in test accuracy by 2.01% and 2.17% on ResNet-20, and 1.89% and 0.86% on ResNet-32, respectively.

Furthermore, both DE and SPL+DE made the models trained by SPL converged, and DE achieved a higher convergence speed by 8 and 7 additional epochs on ResNet-20 and ResNet-32, respectively. Note that applying SPL alone could not make the two models converged (up to using 250 epochs) and adding DE either training the resultant models alone with DE or training them with DE and SPL together. The result shows DE addressing the inability of SPL to converge a model and demonstrates the complementary nature of DE and SPL.

A clear trend can be found in Fig. 4. Fig. 4 can be read similarly as Fig. 3. We observed that the training and test accuracies of these models trained by SPL fluctuated during the training process. Furthermore, the models did not converge after incorporating SPL into the training purpose. DE and SPL+DE presented a more stable and smoother trend in training DL models.

The overall experimental results point out that DE could significantly accelerate the convergence of the models trained by SPL and improve the training and test accuracies of these models at the same time.

C. Threats to Validity

Owing to the inherent imprecision of DL models, the training and test accuracies usually fluctuate during the training process. Besides, some training samples may be "conflicting" with each other, e.g., providing opposite search directions of gradients, which may affect the effectiveness of a method applied in the experiment. It could be the underlying reason for requiring significantly different numbers of epochs for model convergence across experimental trials. To alleviate this threat, we ran the experiment five times to present the average results.

We classified models into two model complexity categories based on whether the gradient vanishing problem is resolved. Smaller models without solving this problem cannot be "deep", and they are referred to as models of low complexity. Models with the gradient vanishing problem solved can scale to many hidden layers, and we referred to them as models of high model complexity. In general, there is no universal criterion to classify every model like what we did in this paper. In our experimental design, we chose models with noticeable differences so that there was no mistake in splitting them into the two classes of model complexity. Readers should interpret the experimental results with care.

We used a common convergence criterion to determine whether a DL model had converged after being trained with a certain number of epochs. Using another convergence criterion may give a different result. Changing the parameter value of the convergence criterion may also lead to different numbers of epoch to converge DL models. We had studied to alter this threshold parameter slightly but did not observe significant differences from the results reported in this study. Since the results were almost identical to the results presented in this paper, for brevity, we did not want to overload readers with almost identical tables. However, the results shown in our plots (in Fig. 3) were invariant to this criterion as the criterion only determines which points in these plots to be used as the model convergence points.

The experiment had compared DE with SPL. The original purpose of SPL is to train a DL model with easier samples ahead of more complex samples. SPL does not aim to accelerate the convergence of DL models during the training process. There was no surprise that SPL did not consistently outperform BL significantly in terms of the number of epochs to converge. DeepEutaxy mainly focuses on fixing the inefficient convergence problem in the training process from the batch prioritization perspective, which makes it different from existing fixing techniques or optimizing approaches that are orthogonal to DeepEutaxy (e.g., each training script enhanced by DeepEutaxy can apply each of these techniques). As we have shown in the experiment, DeepEutaxy can be applied on top of existing techniques and produce more promising results. This finding is promising.

It is still worth generalizing the idea of DeepEutaxy to compare it to studies in other research areas like test case prioritization for the software testing purpose [4][11][50] that apparently shares the similarity of reordering elements.

However, we note that our design of prioritization on batches is a direct consequence of guiding DL models to train their weights on the "areas" requiring more attention first. These "areas" are modeled as the notion of search direction in our work, and this notion is represented in a differential term over the weights of a DL model before and after training on the corresponding batch. In general, there are many ways to model these "areas" and notions with concrete representations. It is interesting to explore them in future work.

We used six model-dataset-optimizer combinations in the experiment. The evaluation on more DL models and datasets with different hyperparameter configurations and optimizers can strengthen the generalization of the experiment further.

The experiment measured the convergence in training accuracies and reported the number of epochs used. However, due to inherent randomness in training a DL model, the convergence criterion might be fulfilled by chance. As shown in Fig. 3, the plots of training accuracy during the training process can provide more detailed information to cross-validate the results. From the results, we did not observe abnormality to point to premature satisfaction of the convergence criterion.

A threat is the implementation bugs in the training scripts and the underlying frameworks. The bugs in certain underlying frameworks have been reported in previous work [33] that they could affect the performance of the DL models under training or their inferences. It is beyond our knowledge to what extent these implementation bugs affect the current results for the implementation of bugs in our DL models. We have used ResNet-20, ResNet3-32, and MobileNetV2, and two CNN models. We have inspected their code and did not observe abnormality.

The experiment evaluated DE in terms of model convergence as well as training and test accuracies and losses. We compared DE with BL and SPL with a variety of model architecture, datasets, and optimizers to evaluate the effectiveness of DE to strengthen the generalization of the experiment.

V. Further Discussion

We have formulated the fixing of the inefficient convergence problem as a kind of fixing to alleviate the performance problem incurred in training a DL model. As we have shown in the experiment, DeepEutaxy could accelerate the convergence of all six DL models significantly compared to BL and SPL.

Our results provide the first piece of empirical evidence that training batches with descending diversity on the weight changes demonstrate a promising direction to guide a DL model to learn *generalized* features from batches leading to more significant changes in weights.

In practice, training a practical DL model on a real-world dataset requires tremendous training efforts. For example, as reported in [12], DL models (i.e., ResNet-18 and ResNet-34) are trained with 600,000 iterations on the ImageNet dataset [36], which may require several days or weeks to obtain a converged model. As we have shown in the experiment, DeepEutaxy has the potentials to reduce such training efforts significantly. It is promising to apply and extend DeepEutaxy or its basic ideas to study and address other challenging problems in DL model training.

When applying SGD, each batch of training dataset provides a search direction in the feature space of parameters. It is challenging to find the most suitable direction towards the global optimum in a single step without prior knowledge. Previous studies [19][24] often look for applying regularization terms in the fixing process. Regularization errors are used in back-propagation to update model weights, which needs the feature maps of the forward-propagation on processing the same batch of samples. Owing to the limitation of GPU memory, feature maps across batches could not be cross-referenced without significant performance overheads. We are also unaware of published work presenting techniques to deal with regularization errors across batches efficiently.

Since previous techniques are unaware of how a DL model performs over different batches, their performances have not been optimized to their full potentials in relation to this unexplored dimension. DeepEutaxy oversees the training process, and because of this, it can take advantage of how errors could be handled with priority by a DL model in the subsequent epochs, which provides a new perspective in process optimization unseen to previous work. The empirical results show that the novel design of DeepEutaxy can lead to significant and positive training effects on models with both high and low model complexities.

The inherent randomness of a DL model may affect the convergence of training the model. For example, using Dropout [42] layers may impact the convergence speed of the model because, in each iteration, a subset of randomly selected weights will be updated while others remain unchanged. The intuitive idea of introducing randomness into DL models is to generalize the model better. We noticed that Dropout had been applied in MobileNetV2, and our experimental result has shown that DeepEutaxy outperformed the baseline with MobileNetV2 and MobileNetV2-2 on the CIFAR-10 dataset, indicating the effectiveness of DeepEutaxy. However, we did not comprehensively quantify the effect of the randomness of DL models on DeepEutaxy, and it is worth exploring it further.

An interesting observation is that DL models trained by DeepEutaxy could achieve 100% in training accuracy earlier than BL and SPL. However, this situation might prevent the model from searching for the global optimum further. That is, suppose the training accuracy is 100%, indicating that all training samples are classified correctly, then the loss would be marginal, and thus the gradients derived from the loss function would be minimal, and the weights of the model would be updated slightly and trapped into local optima. There are various proposals to alleviate such problems by adding regularization terms [19] or exquisitely crafted loss functions [47], or using well-designed model architectures [42]. It is interesting to explore the performance of those techniques combining with DeepEutaxy on training DL models.

In practice, it may be computationally expensive to calculate the gradients for all batches and mutually compare their similarity. It may be more practical to choose a subset of weights and gradients and observe their behaviors. Some studies [1][20] have proposed to use the states of merely one

layer, e.g., the layer right before the output *softmax* layer, to approximate both the features and differences among the status of DL models. Such an approximation may help reduce the computational cost involved in model training. In our experiment, all layers are involved in calculating the similarity between batches, and we did not explore the above possibility further.

DeepEutaxy applied the cosine similarity to measure the diversity between weights and gradients. The cosine similarity has been widely applied in deep learning [25][46]. It measures how similar these two objects are irrespective of their sizes. In this work, we measure to what extent the gradients can provide "new" information for updating weights of a DL model. As we have mentioned, if weights and gradients have the same direction, the update will enhance the existing features obtained only. Other similarity measurements like Euclidean distance may not be suitable in this case. It is interesting to explore other possible distance functions. However, the change in the similarity measure will not change the basic idea of DeepEutaxy. It seems to us that the above-mentioned experimental procedure will still be applicable to evaluate such DeepEutaxy variants.

For the implementation of SPL, we noticed that the performance of SPL could be sensitive to λ and α in Equations (3) and (4). If λ decreases toward 0, SPL will be increasingly degraded toward BL. If λ is too large, then all samples will be identified as difficult, and the model will not be trained, which indicates that developers need to fine-tune hyperparameters carefully when applying SPL into practice. DeepEutaxy does not need such fine-tuning. Having said that, in our experiment, we have demonstrated that DeepEutaxy can further improve the performance of a model trained by SPL.

VI. Related Work

A. Acceleration

Song et al. [41] proposed to use three steps to compress neural networks. First, essential neurons and weights are learned and recognized, while the rest are pruned. Then, weights are quantized, and each weight is stored in fewer bits. Weight sharing is also implemented where weights that have similar values are assigned to the same number in order to save memory. After that, the model with the remaining connections is retrained. The proposed method can reduce the size and complexity of neural networks significantly, while retraining neural networks may increase extra time consumption in terms of neural network acceleration.

Some studies examine DL optimization inspired by the learning process of humans and animals. Bengio et al. [3] proposed curriculum learning to optimize neural networks. The basic idea is to introduce samples that are easy to learn at the beginning, and they gradually increase the complexity of input data for training. The difficulty of training samples is determined by a predefined rank function. The experimental results showed that curriculum learning has an impact on the speed of convergence, while the performance of curriculum learning depends on the quality of the rank function heavily.

Kumar et al. [20] consider the learning process as a joint optimization problem by introducing sample weights as a regularization term to the loss function and proposed Self-Paced Learning (SPL). In each iteration of the training process, the weights of each sample, as well as a parameter vector, are adjusted and updated. In that case, the difficulty of samples is defined and self-learned by the DL model itself. Besides, after the training process, a sample that is difficult at the beginning would become more manageable in retrospect.

Jiang et al. [13] studied the relationship between curriculum learning and self-paced learning and proposed self-paced curriculum learning (SPCL). SPCL considers both prior knowledge and the runtime information by introducing a regularization term composed of sample weights that are determined by the predefined curriculum to the loss function. SPCL can be regarded as a generalization of SPL.

B. Testing and Debugging

The objective of DL testing is to evaluate the quality and expose defects of neural networks. Recent studies proposed various test criteria to evaluate test sufficiency. Pei et al. [32] proposed to use neuron coverage to measure test sufficiency. Given a test dataset, a neuron is said to be covered if the output of the neuron is higher than a specified threshold at least once. Similar to neuron coverage, Ma et al. [27] proposed a more refined way to measure neuron coverage that further divides the output value of neurons into several sections, and a neuron is said to be covered if all sections are covered.

In the area of debugging and fixing neural networks, Ma et al. [29] proposed MODE, a neural network debugging method to repair potential defects of neural networks. Differential analysis is implemented to locate which neurons or layers are problematic, and a feature map is generated to identify the perception of DL models. To debug the DL models and fix bugs, an input selection method is implemented by utilizing a feature map, and the DL model is retrained based on the selected input data.

Tao et al. [44] proposed TRADER to debug recurrent neural networks. To address the embedding problem of the RNN model, a divergence analysis is implemented to investigate the buggy embedding behaviors. Given an input, the trace of the model is obtained, i.e., the intermediate states of the model are collected. To fix these buggy traces, the model is trained on buggy inputs by adding noises to the intermediate states. The experimental results showed that TRADER could consistently increase the accuracy by 5.37% on average.

Zhang and Chan [49] proposed Apricot to fix deep learning models. The intuition is that the DL model that is trained on a small subset of the original training dataset may contain essential information for the correct classification of some particular inputs. Apricot firstly generates a set of such DL models and then utilizes them to provide search directions in adjusting the weights of the DL model under repair.

VII. Conclusion

In this paper, we have proposed DeepEutaxy. To the best of our knowledge, it is the first work to address the inefficient

convergence problem of DL models in the training process by batch prioritization. The basic idea of DeepEutaxy is to first train a DL model with the training dataset with several epochs, followed by measuring the diversity between weights of the DL model and the gradients with respect to a given batch before and after the training on that batch and finally reordering. The batch sequence is dynamically reordered based on their measured diversity in descending order. At the point of measurement, batches with larger diversities are fed into the model with higher priority. The batch sequence is re-prioritized after a certain number of epochs. We have evaluated DeepEutaxy with six DL models on the MNIST and CIFAR-10 datasets with different optimizers, showing that DeepEutaxy is particularly promising in fixing the convergence problem in training DL models with different model complexity.

Our future work will mainly focus on investigating the relationship between the training batches and the performance of the DL model in a more refined manner. There are some challenging problems, such as understanding the internal characteristics of neural networks and updating weights with fewer training steps with manageable training and test losses. It is worth exploring the intermediate behaviors of DL models that might be useful for their efficient model optimization. We would like to study these issues in the future. We are also aware that not all gradients of weights are required to observe the internal behaviors of DL models. To reduce the computational cost, weights that are more representative can be chosen for investigation.

REFERENCES

[1] I. A. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design and application," Journal of microbiological methods, vol. 43, no. 1, pp. 3-31, 2000.

[2] J. Banzi, I. Bulugu, and Z. Ye, "Learning a deep predictive coding network for a semi-supervised 3D-hand pose estimation," IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 5, pp. 1371-139, 2020.

[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," In Proc. 26th international conference on machine learning, 2009, pp. 41-48.

[4] T. Byun, V. Sharma, A. Vijayakumar, et al. "Input Prioritization for Testing Neural Networks," In Proc. IEEE International Conference on Artificial Intelligence Testing (AITest'19), 2019, pp. 63-70.

[5] Convolutional Neural Networks for CIFAR-10, https://github.com/BIGBALLON/cifar-10-cnn.

[6] CyclicLR, https://github.com/Jie-Yuan/CyclicLR.

[7] CyclicLR Scheduler PyTorch, https://github.com/automan000/CyclicLR_Scheduler_PyTorch.

[8] DeepEutaxy, https://github.com/DemoAuguste/DeepEutaxy.

[9] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," Journal of machine learning research, vol. 12, no. 7, 2011.

[10] H. Eniser, S. Gerasimou, and A. Sen, "DeepFault: Fault Localization for Deep Neural Networks," In Proc. International Conference on Fundamental Approaches to Software Engineering (FASE'19), 2019, pp. 171-191.

[11] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "DeepGini: Prioritizing Massive Tests to Enhance the Robustness of Deep Neural Networks," arXiv: 1903.00661, 2020.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," In Proc. IEEE conference on computer vision and pattern recognition, 2016, pp.770-778.

[13] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. Hauptmann, "Self-paced curriculum learning," In Proc. AAAI conference on artificial intelligence, 2015.

[14] L. Jiang, D. Meng, S. Yu, Z. Lan, S. Shan, and A. Hauptmann, "Self-paced learning with diversity," In advances in neural information processing systems, 2014, pp. 2078-2086.

[15] P. Kebria, A. Khosravi, S. Salaken, and S. Nahavandi, "Deep Imitation Learning for Autonomous Vehicles Based on Convolutional Neural Networks," IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 1, pp. 82-95, 2019.

[16] Keras: the Python deep learning API, https://keras.io/.

[17] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems (NIPS'12), 2012, pp. 1097-1105.

[19] J. Kukacka, V. Golkov, and D. Cremers, "Regularization for Deep Learning: A Taxonomy," arXiv: 1710.10686, 2017.

[20] M. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," In advances in neural information processing systems, 2010, pp. 1189-1197.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol.521, pp.436-444, 2015.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," In Proc. Of the IEEE, 1998, pp. 2278-2324.

[23] H. Liu, I. Chatterjee, M. Zhou, X. Lu, and A. Abusorrah, "Aspect-Based Sentiment Analysis: A Survey of Deep Learning Methods," IEEE Transactions on Computational Social Systems, vol. 7, no. 6, pp. 1358-1375, 2020.

[24] C. Louizos, M. Welling, and D. P. Kingma, "Learning Sparse Neural Networks through L_0 Regularization," arXiv: 1712.01312, Jun. 2018.

[25] C. Luo, J. Zhan, X. Xue, L. Wang, R. Ren, and Q. Yang, "Cosine Normalization: Using Cosine Similarity Instead of Dot Product in Neural Networks," In Proc. Of The International Conference on Artificial Neural Networks (ICANN'18), 2018, pp. 382-391.

[26] L. Ma, F. Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, "DeepCT: Tomographic combinatorial testing for deep learning systems," In Proc. IEEE conference on software analysis, evolution and reengineering, 2019, pp. 614-618.

[27] L. Ma, F. Juefei-Xu, F. Zhang, et al., "Deepgauge: Multi-granularity testing criteria for deep learning systems," In Proc. ACM/IEEE International Conference on Automated Software Engineering, 2018, pp.120-131.

[28] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, and F. Xu, "DeepMutation: mutation testing of deep learning systems," In Proc. IEEE conference on software reliability engineering, 2018, pp. 100-111.

[29] S. Ma, Y. Liu, and W. Lee, "MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection," In Proc. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 175-186.

[30] MobileNetV2, https://github.com/xiaochus/MobileNetV2.

[31] THE MNIST DATABASE of handwritten digits, http://yann.lecun.com/exdb/mnist/.

[32] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated Whitebox Testing of Deep Learning Systems," In Proc. Symposium on Operating Systems Principles '17, 2017, pp.1-18.

[33] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, "CRADLE: Cross-Backend Validation to Detect and Localize Bugs in Deep Learning Libraries," In Proc. 41st International Conference on Software Engineering (ICSE'19), 2019, pp. 1027-1038.

[34] M. Popel, M. Tomkova, J. Tomek, L. Kaiser, J. Uszkoreit, O. Bojar, and Z. Žabokrtský, "Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals," Nature Communications, vol. 11, no. 1, pp. 1-15, 2020.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by error propagation," Parallel Distributed Processing:

Explorations in the Microstructure of Cognition: Foundations, 1987, pp. 318-362.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, et al., "ImageNet Large Scale Visual Recognition Challenge," International Journal of Computer Vision, vol. 115, no. 3, pp. 211-252, 2015.

[37] M. Sandler, A. Howard, M. Zhu, et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks," arXiv: 1801.04381, Mar 2019.

[38] Self-Paced Learning for Machine Learning, https://towardsdatascience.com/self-paced-learning-for-machine-learning-f1c489316c61.

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556v6 [cs], Apr. 2015.

[40] L. Smith, "Cyclical Learning Rates for Training Neural Networks," arXiv: 1506.01186, Apr. 2017.

[41] H. Song, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," arXiv: arXiv:1510.00149, 2015.

[42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929-1958, 2014.

[43] Y. Sun, M. Wu, W. Ruan, et al. "Concolic Testing for Deep Neural Networks," In Proc. The 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE'19), 2018, pp. 109-119.

[44] G. Tao, S. Ma, Y. Liu, Q. Xu, and X. Zhang, "TRADER: Trace Divergence Analysis and Embedding Regulation for Debugging Recurrent Neural Networks," In Proc. IEEE/ACM International Conference on Software Engineering (ICSE'20), 2020.

[45] TensorFlow: An end-to-end open-source machine learning platform, https://www.tensorflow.org/.

[46] N. Wojke and A. Bewley, "Deep Cosine Metric Learning for Person Re-identification," In Proc. IEEE Winter Conference on Applications of Computer Vision (WACV'18), 2018, pp. 748-756.

[47] C. Xu, C. Lu, X. Liang, J. Gao, W. Zheng, T. Wang, and S. Yan, "Multi-loss Regularized Deep Neural Network," IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, no. 12, pp. 2273-2283, 2015.

[48] M. Zeiler, "ADADELTA: an adaptive learning rate method," arXiv: arXiv: 1212.5701, 2012.

[49] H. Zhang and W.K. Chan, "Apricot: a weight-adaptation approach to fixing deep learning models," In Proc. IEEE/ACM International Conference on Automated Software Engineering (ASE'19), 2019, pp. 376-387.

[50] L. Zhang, X. Sun, Y. Li, and Z. Zhang, "A Noise-Sensitivity-Analysis-Based Test Prioritization Technique for Deep Neural Networks," arXiv: 1901.00054, Jan. 2019.

Hao Zhang received the B.Eng. and M.Sc. degrees from Beihang University, China, in 2015 and 2018, respectively. He is currently a Ph.D. student of computer science at City University of Hong Kong. His main research interests are the interplay between software engineering and artificial intelligence, as well as program debugging. His work has been reported in ASE and TOSEM.

W.K. Chan. (M'04) received the B.Eng., M.Phil., and Ph.D. degrees from The University of Hong Kong, Hong Kong. He is currently an Associate Professor with the Department of Computer Science, City University of Hong Kong. His research results have been reported in more than 100 major international journals and conferences. His current research interests include addressing the software testing and program analysis challenges faced by developing and operating large-scale applications and deep learning systems. Dr. Chan serves on the editorial boards of the Journal of Systems and Software, Software Testing, Verification and Reliability, and International Journal of Creative Computing. He has served as PC/RC members of many international conferences and program co-chairs of several conferences and workshops.