The 36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021) 15-19 November, 2021

DEVIATE: A Deep Learning Variance Testing Framework

Hung Viet Pham
University of Waterloo
hvpham@uwaterloo.ca

Mijung Kim
Purdue University[1]
mijungk@unist.ac.kr

Lin Tan
Purdue University
lintan@purdue.edu

Yaoliang Yu
University of Waterloo
yaoliang.yu@uwaterloo.ca

Nachiappan Nagappan
Microsoft Research[2]
nnagappan@acm.org

[1]*This work was done when Mijung was at Purdue University. She is currently with Ulsan National Institute of Science and Technology.*
[2]*This work was done when Nachiappan was with Microsoft. He is currently with Facebook.*

***Abstract*—Deep learning (DL) training is nondeterministic and such nondeterminism was shown to cause significant variance of model accuracy (up to 10.8%). Such variance may affect the validity of the comparison of newly proposed DL techniques with baselines. To ensure such validity, DL researchers and practitioners must replicate their experiments multiple times with identical settings to quantify the variance of the proposed approaches and baselines. Replicating and measuring DL variances reliably and efficiently is challenging and understudied.**

We propose a ready-to-deploy framework DEVIATE that (1) measures DL training variance of a DL model with minimal manual efforts, and (2) provides statistical tests of both accuracy and variance. Specifically, DEVIATE *automatically* analyzes the DL training code and extracts monitored important metrics (such as accuracy and loss). In addition, DEVIATE performs popular statistical tests and provides users with a report of statistical p-values and effect sizes along with various confidence levels when comparing to selected baselines.

We demonstrate the effectiveness of DEVIATE by performing case studies with adversarial training. Specifically, for an adversarial training process that uses the Fast Gradient Signed Method to generate adversarial examples as the training data, DEVIATE measures a max difference of accuracy among 8 identical training runs with fixed random seeds to be up to 5.1%.

Tool and demo links: https://github.com/lin-tan/DEVIATE

***Index Terms*—deep learning, variance, nondeterminism**

I. Introduction

With the increasing availability of large datasets and parallel processing power, deep learning (DL) systems have demonstrated their capability in challenging tasks such as image processing [1], speech recognition [2], and natural language processing [3]. Recently, DL systems are the key components of safety-critical systems such as air traffic control [4], autonomous vehicle [5], medical diagnostics [6], and malicious code detector [7]. To improve model accuracy and training efficiency for such DL systems, nondeterminism is used during the training process such as the shuffling the order of training data batch to prevent overfitting and speed up training [8]. Due to this nondeterminism, performing the training multiple times with an identical setting (i.e., same hyperparameters, same GPU, same libraries, etc.) produces models with significantly different accuracy [9].

The nondeterminism can be introduced by algorithmic nondeterminism-introducing (NI)-factors and implementation-level NI-factors. Algorithmic NI-factors are intentionally introduced random processes (e.g., random seeds) that help improve training efficiency and model accuracy. Implementation-level NI factors are the optimization of DL algorithms on parallel computing hardware (e.g., GPU) in DL libraries such as TensorFlow, Pytorch, and cuDNN.

Recent research recognizes and quantifies the variance in DL training [10], [9] by showing that both the algorithmic and implementation-level NI-factors cause a significant variance of DL system accuracy and training time between identical training runs. This variance potentially could affect the validity of the improvement claimed by newly proposed techniques in the DL research community. DL researchers and practitioners must perform their DL training experiment multiple times with identical settings to measure the variance of their proposed approaches and then compute the statistical test when comparing to the baseline approaches. With the tested results, DL practitioners can make informed choices of techniques when applying DL properly and effectively in their applications.

However, consistently replicating and measuring the experimental variances requires additional effort which could be one of the reasons why measuring the variance of the proposed techniques is often overlooked by prior work as reported by a prior study [9]. For example, to measure the variance, developers need to organize and perform identical runs multiple times and make sure that the experiment is correctly set up so that the runs are as identical with each other as possible. In addition, once the data from the multiple runs are collected, it is nontrivial to consistently select and perform the correct statistical tests.

To aid with the replication and improvement of the validation of research, we propose DEVIATE that (1) consistently measures the variance with minimum user efforts, and (2) provides appropriate statistical tests. DEVIATE replicates multiple training runs with little to no input from the users.

DEVIATE *automatically* analyzes the DL system source code, extracts important metrics (such as accuracy, loss...) that are monitored by the authors, and stores those metrics in a consistent format. DEVIATE performs popular statistical tests (such as confidence interval of standard deviation, Mann–Whitney U-test, and Cohen's d effect size) and provides users with a report of how well the proposed technique performs in comparison to the selected baselines.

To demonstrate the effectiveness of DEVIATE, we perform a case study with adversarial training [11]. We found that

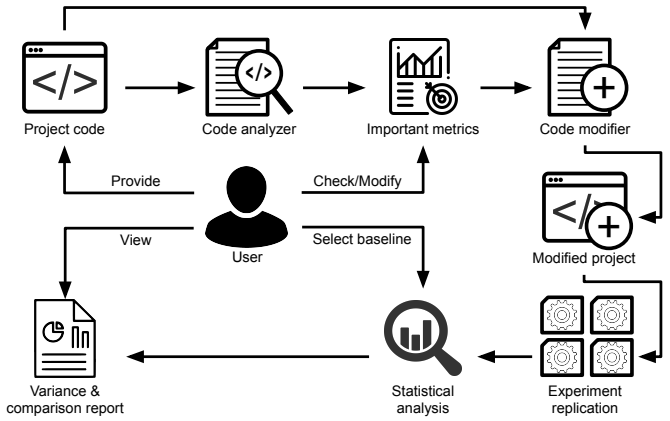

Fig. 1: The overview of DEVIATE.

Fig. 2: Example of metrics (`nat_acc` and `adv_acc`) that DEVIATE monitors.

```
1  from dl_logging_helper import DLVarLogger
2  for ii in range(max_num_training_steps):
3    DLVarLogger.beginLoop(ii)
4    ...
5    if ((ii % num_output_steps) == 0):
6      nat_acc = run(model.accuracy, dict=nat_dict)
7      adv_acc = run(model.accuracy, dict=adv_dict)
8      print('Step {}:({})'.format(ii, datetime.now()))
9      print('nat accuracy {:.4}%'.format(nat_acc*100))
10     DLVarLogger.log('nat_acc*100', (nat_acc*100))
11     print('adv accuracy {:.4}%'.format(adv_acc*100))
12     DLVarLogger.log('adv_acc*100', (adv_acc * 100))
13   ...
14   DLVarLogger.endLoop()
15 ...
16 DLVarLogger.endLogger()
```

apart from the standard training approaches, other training approaches can also have large variance. For example, with the standard training procedure, Pham *et al.* report a difference up to 1.9% in accuracy among 16 identical training runs of ResNet56 network with fixed random seeds. However, with a more complex adversarial training process [11] that uses Fast Gradient Signed Method (FGSM) [12], DEVIATE measures a difference of robustness (accuracy on adversarial samples) and effectiveness (accuracy on natural samples) up to 5.1% and 2.0% in accuracy, respectively, among only 8 identical training runs with fixed random seeds.

II. DEVIATE

While replicating identical DL experiments and measuring the variance of them may appear to be straightforward, this task has many research and engineering challenges. First, it requires extra efforts to automatically monitor important metrics (e.g., epoch, accuracy, and loss) in training source code and record them in an organized manner among identical experiment runs. Since different users tend to use different coding styles in writing source code (e.g., a simple `for` loop or a dedicated method), automatically identifying the code location that monitors training metrics is challenging. Second, once the results from multiple experiment runs for both the proposed approach and baselines are ready, it is nontrivial to know which statistical tests to use for analyzing the variance among multiple runs of individual approaches and evaluating the improvements over baseline approaches with the variance taken into account.

We introduce DEVIATE framework that addresses these challenges. Figure 1 shows the overview of DEVIATE. The *code analyzer* automatically processes the training source code and extract the variables where important metrics such as epoch, accuracy, and loss, are assigned. DEVIATE supports an option for users to double-check the extracted metrics and select only the relevant metrics if necessary. The *code modifier* then injects the logging code into the original source code to record these metrics. DEVIATE then systematically replicates the experiments multiple times with the most identical setting possible. Once the experiments are completed, DEVIATE performs statistical tests to analyze the variance of the user's proposed approach as well as the comparison of the proposed approaches with baselines. Hence, DEVIATE enables the reproducibility of the proposed approach and ensures the validity of comparison results.

Metric extraction: To measure the variance of DL training, DEVIATE first needs to know what metrics to monitor and record. DEVIATE analyzes user code and extracts variables that store important metrics which measure the quality of the model (e.g., accuracy) or the speed of the training process (e.g., training time).

To design an automatic source code analyzer, we first collect and manually analyze 52 research projects from published papers that propose new DL approaches. These projects contain the source code of DL experiments ranging from testing new DL architectures to DL model compression to Neural Architecture Search (NAS). From these projects, we observe several heuristics that help DEVIATE correctly analyze and modify source code.

The authors of the source code often monitor and log the important metrics (e.g., loss or accuracy) in their code (i.e., log to a file or print to the screen). DEVIATE applies this hypothesis and extracts variables from the `log` and `print` statements. However, not all variables in these statements are relevant, sometimes the authors print out the runs' information (e.g., model identifiers, training optimizer names) or the settings (e.g., learning rates) which are not as important as the metrics that measure the quality of the model. DEVIATE applies another heuristic by selecting metrics that are formatted as high precision floating numbers (i.e., with 4 or more decimals) as the authors want to record important metrics as accurately as possible. For example, Figure 2 presents snippet of adversarial training code for ResNet32 and WRN32-10 networks. Important metrics such as natural accuracy and adversarial accuracy are assigned to variables `nat_acc` and `adv_acc` (lines 5 and 6), respectively. The values of these variables are printed using `print` statements (lines 8 and 10) with high precision at 4 decimals.

DL model training is an iterative process where the model is initialized (e.g., with random initial weights) and updated at each iteration using the training data until a stopping criterion is met (e.g., reached a certain loss threshold or a fixed number of iterations is done). This iterative process is often implemented as the main training loop with a counting variable indicating the current iteration of the training process. After each execution of this loop, the quality of the model changes so the authors often record the quality and timing metrics along with the counting variable within this loop at some logging interval (i.e., skipping some iteration for better efficiency). DEVIATE uses this observation to detect the loop and count variable by detecting the looping variable that is logged or printed along with other metrics. Once the main loop is detected, the logged or printed metrics within this loop are considered important measurements of the model quality and model training time. In Figure 2, DEVIATE detects the main loop (line 2) that contains `print` statements of the looping variable `ii` along with important metrics for the model quality (i.e., accuracy).

DEVIATE utilizes these heuristics to archive a high extraction accuracy (98.7% – Section III) but since it is not 100%, it provides users the options to double-check and, if necessary, modify the list of extracted important variables. Once the user confirms the list, DEVIATE injects logging statements that will systematically log the selected variables in a consistent format. For example, Figure 2 shows the injected logging statements (lines 9 and 11) as well as the injected marking statements (lines 2 and 13) that indicate the beginning and the end of the training loop to monitor each iteration.

Variance measurements and statistical tests: For each selected metric, DEVIATE records the value for each iteration. The user can select an aggregate strategy for each metric. For example, when analyzing the validation accuracy of a model, the user can choose to analyze the best accuracy among all iterations. Alternatively, the user can choose to analyze the validation accuracy of the last iteration. For some metrics, such as running time, the user can choose to analyze the average running time across all iterations.

For each set of identical experiments, DEVIATE computes the mean, the standard deviation, the coefficient of variation (i.e., the ratio of the standard deviation over mean), the maximum and minimum values as well as the sample range (i.e., the difference between maximum and minimum values) of the aggregated values across all identical runs. Since the range of the recorded metrics can be very different, DEVIATE then sorts the selected metrics using the coefficient of variation. The coefficient of variation presents the mean normalized standard deviation of the selected metrics and can be compared across different metrics.

DEVIATE provides several statistical tests to assess the similarity of sample distribution such as the Student T-test, the Mann Whitney U-test. DEVIATE reports the p-value of the test as well as the effect size (i.e., computed as the Cohen's d [13]). To compare two sets of identical experiments, DEVIATE performs statistical tests between pairs of matched metrics (i.e., with the same name or matched by users).

III. Evaluation

In this section, we test the accuracy of DEVIATE's metric extraction module against a benchmark of 21 projects for techniques that improve DL efficiency. We also demonstrate DEVIATE's usefulness in quantifying the variance of DL system training by applying DEVIATE to a DL project.

(A) Metric extraction performance: One of the main features of DEVIATE is its ability to detect metrics to monitor (e.g., accuracy, loss, time, etc.). Since we apply several heuristics to detect these metrics, we want to test how well DEVIATE extracts such metrics in a wide range of research source code. To do this, we collect a set of 54 research projects from papers that focus on efficient DL such as distillation, quantization, model pruning, efficient DL architecture, and neural architecture search (NAS). Out of these, we select 21 projects that use Python, clearly documented so that we can identify the main training files. The reason that we only focus on projects that are written in Python is that Python is the most popular language used for machine learning [14]. DEVIATE requires the main training file to be provided for each experiment because automatically detect such a file is nontrivial and remains as future work.

We prepare the extracted main training files to be used as a metric detection test dataset by manually investigating each file to identify the metrics that indicate the performance of the model. These metrics could be general metrics such as accuracy, loss, and time. We also identify metrics specific to each approach (e.g., compression rate for pruning techniques). All manual inspection is done by two authors independently and any disagreements are resolved in the end.

We apply DEVIATE metric extraction module on the metric test data to see how well our extraction heuristics are working. DEVIATE fails to extract metrics from 7 projects due to training code spread across multiple source code files and bespoke metric logging code. Supporting such projects remains as future work. DEVIATE successfully extracts the main loop for all other 14 projects and extracts 153 variables with an accuracy of 98.7%. Overall, DEVIATE has good accuracy in extracting the main training loops and metrics. DEVIATE also provides users with options to remove incorrect or add missing metrics in case the automatically extract metrics do not meet the user requirements.

(B) Adversarial training case study: Recently, *adversarial training*, which focuses on improving the robustness of the DL system training has become popular. One of the first adversarial training approaches [11] trains robust DL models directly on adversarial example instances. Such work claims robustness of resulted models but their evaluation fails to take into account the variance of the DL system. We investigate how the variance in DL training affects the robustness of

TABLE I: The accuracy (%) difference between adversarial training runs

Network	Training							Test						
	acc_{adv}				acc_{nat}			acc_{adv}					acc_{nat}	
	Attack	Diff	SDev	Avg	Diff	SDev	Avg	Row	Attack	Diff	SDev	Avg	Diff SDev	Avg
ResNet32	FGSM	**4.0**	1.5	97.6	**3.9**	1.5	93.0	1	FGSM	**5.1**	1.7	93.6	**2.0** 0.7	87.8
								2	PGD	<0.1	<0.1	<0.1		
WRN32-10	FGSM	0.4	0.1	99.5	0.9	0.3	96.6	3	FGSM	1.1	0.4	96.2	1.1 0.3	91.4
								4	PGD	<0.1	<0.1	<0.1		

the adversarial training process by applying DEVIATE on the source code[1] provided by the authors.

Specifically, the adversarial training process trains a network (e.g., ResNet32 or WideResNet32-10) from scratch on the CIFAR10 dataset using only adversarial examples generated from adversarial attacks (e.g., Fast Gradient Signed Method—FGSM [12] or Projected Gradient Descent—PGD [11]). The trained models are evaluated by both their effectiveness (accuracy on the unseen natural (i.e., *benign*) input and their robustness (accuracy on the unseen *adversarial* example input). In this case study, we investigate the accuracy variance of both natural input (acc_{nat}) and adversarial input (acc_{adv}) during training and inference (i.e., test) among replicated runs. *We disable all algorithm NI-factors and using the fixed random seed provided by the authors.* Table I shows the maximum difference (*Diff*), standard deviation (*SDev*), and the average (*Avg*) of the *Training* and *Test* accuracy of the adversarial input (acc_{adv}) and normal input (acc_{nat}). The columns *Attack* denote the method that generates the adversarial examples.

DEVIATE successfully detects the two metrics (i.e., adversarial example accuracy—acc_{adv} and natural sample accuracy acc_{nat}) in both training and test code, adds monitoring code, performs 32 fixed-seed identical training runs for four training configurations (8 runs each), and presents the accuracy variance of DL adversarial training automatically. In addition, DEVIATE presents p-values of comparison of pairs of network and attack technique combination.

1) Large accuracy differences: The variance of test acc_{nat} and test acc_{adv} reveals the stability of the trained models when applied in the real world. In particular, ResNet32 models trained with FGSM attack have test acc_{adv} (i.e., adversarial robustness) difference of 5.1% (when test using FGSM attack, shown on row 1). The same training configuration also has a large test acc_{nat} (i.e., effectiveness) difference of 2.0%. Since the main focus of adversarial training is increasing the adversarial robustness while also maintaining the effectiveness of the trained models, such large differences in both categories should be taken into consideration by DL practitioners. Our result demonstrates that it is nontrivial to estimate such variance and shows the importance of DEVIATE.

2) Impact of variance on valid comparison: We use the two training configurations using FGSM attacks to demonstrate DEVIATE's capacity in helping DL researchers and practitioners compare different training configurations. Table I shows that between the two configurations using FGSM attack (rows 1 and 3), the one with the WRN32-10 network has better adversarial robustness against white-box FGSM attack with an average test acc_{adv} of 96.2% (comparing to average robustness of 93.6% of ResNet32 models). DEVIATE computes the Mann–Whitney U-test which confirms that WRN32-10 models are significantly (p-value of < 0.05) more robust than ResNet32 with a Cohen's d effect size of 3 (which is a very large effect). However, in the extreme case, where each configuration is only run once, the difference in robustness could be just 0.2% between 95.6% (the most robust ResNet32 model) and 95.8% (the least robust WRN32-10 model). This example shows the importance of DEVIATE's replication runs and the statistical test in making valid comparisons and conclusions about the improvement of DL approaches.

[1]https://github.com/MadryLab/cifar10_challenge

IV. Related Work

The most relevant work is a recent study that quantifies the variance caused by nondeterministic factors in deep learning training from scratch [9]. Our work differs as DEVIATE is a ready-to-deploy framework that replicates DL experiments with minimal user effort and also automatically generates a variance report for a newly proposed approach as well as comparisons with other approaches. We also study an additional training approach (i.e., adversarial training), which is not studied in the prior work. Other than this, our work primarily relates to the work that provides benchmarks for DL hardware [15] and DL frameworks [16], [17], [18], [19], [20], [21]. None of the benchmarking work considers the impact of NI-factors and the variance among multiple identical training runs like our work.

V. Conclusions

Deep learning (DL) training is nondeterministic which both the DL algorithm and DL software implementation cause significant a variance of model accuracy. Such variance may affect the validity of the comparison results of newly proposed DL techniques with baselines. To ensure such validity, DL researchers and practitioners must replicate their experiments multiple times with identical settings to quantify the variance of the proposed approaches and baselines. We propose DEVIATE that (1) measures DL training variance of a DL model with minimal manual efforts, and (2) provides statistical tests of both accuracy and variance.

REFERENCES

[1] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *CVPR*, 2012.

[2] G. Hinton, L. Deng, D. Yu, G. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *Signal Processing Magazine*, 2012.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014.

[4] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *DASC*, 2016.

[5] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," in *CoRR*, 2016.

[6] G. J. S. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Med. Image Anal.*, 2017.

[7] Z. Cui, F. Xue, X. Cai, Y. Cao, G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Trans Industr Inform*, 2018.

[8] J. Haochen and S. Sra, "Random shuffling beats SGD after finite epochs," in *ICML*, 2019.

[9] H. V. Pham, , S. Qian, J. Wang, T. Lutellier, J. Rosenthal, L. Tan, Y. Yu, and N. Nagappan, "Problems and opportunities in training deep-learningsoftware systems: An analysis of variance," in *ASE*, 2020.

[10] P. Nagarajan, G. Warnell, and P. Stone, "Deterministic implementations for reproducibility in deep reinforcement learning," in *AAAI Workshop on Reproducible AI*, 2019.

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.

[12] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR*, 2015.

[13] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 1988.

[14] C. Voskoglou, "What is the best programming language for machine learning?" https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7, 2017.

[15] S. Shams, R. Platania, K. Lee, and S.-J. Park, "Evaluation of deep learning frameworks over different hpc architectures," in *ICDCS*, 2017.

[16] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms," in *ASE*, 2019.

[17] V. Kovalev, A. Kalinovsky, and S. Kovalev, "Deep learning with theano, torch, caffe, tensorflow, and deeplearning4j: Which one is the best in speed and accuracy?" 2016.

[18] A. Shatnawi, G. Al-Bdour, R. Al-Qurran, and M. Al-Ayyoub, "A comparative study of open source deep learning frameworks," in *ICICS*, 2018.

[19] L. Liu, Y. Wu, W. Wei, W. Cao, S. Sahin, and Q. Zhang, "Benchmarking deep learning frameworks: Design considerations, metrics and beyond," in *ICDCS*, 2018.

[20] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *CCBD*, 2016.

[21] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Jayarajan, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "Benchmarking and analyzing deep neural network training," in *IISWC*, 2018.