A Tale of Two Cities: Data and Configuration Variances in Robust Deep Learning

Guanqin Zhang, *University of New South Wales, Sydney, 2052, Australia*

Jiankun Sun, *University of Science and Technology, Beijing, 100083, China*

Feng Xu and Yulei Sui, *University of New South Wales, Sydney, 2052, Australia*

H.M.N. Dilum Bandara and Shiping Chen, *Commonwealth Scientific and Industrial Research Organisation, Sydney, 2015, Australia*

Tim Menzies, *North Carolina State University, Raleigh, NC, 27695, USA*

Deep neural networks (DNNs) have widespread applications in industries such as image recognition, supply chain, medical diagnosis, and autonomous driving. However, previous work has shown that the high accuracy of a DNN model does not imply high robustness (i.e., consistent performances on new and future datasets) because the input data and external environment (e.g., software and model configurations) for a deployed model are constantly changing. Therefore, ensuring robustness is crucial to enhance business and consumer confidence. Previous research focuses mostly on the data aspect of model variance. This article takes a holistic view of DNN robustness by summarizing the issues related to both data and software configuration variances. We also present a predictive framework using search-based optimization to generate representative variances for robust learning, considering data and configurations.

Deep neural networks (DNNs) are increasingly deployed into critical systems to solve complex applications, including medical diagnosis,[1] electricity supply chain,[2] and drought resilience.[3] However, despite their wide adoption in many complicated tasks, DNNs are still far from perfect. Existing learning models often yield imprecise or incorrect outputs for real-world applications due to imperfect data and/or configurations.[4] For example, multiple identical training procedures may generate different models with accuracy variances in the presence of various factors including data perturbations (e.g., limited, weakly labeled, and concept-drifting training samples) and variances caused by software implementation and configurations [e.g., nondeterministic deep learning (DL) layers, and random-weight initialization and floating-point imprecision]. Analyzing DNN robustness issues can help practitioners and researchers build more reliable DL systems.

1089-7801 © 2023 IEEE
Digital Object Identifier 10.1109/MIC.2023.3322283
Date of publication 6 October 2023; date of current version 6 November 2023.

Figure 1 shows the experiments on a range of applications, with performance fluctuations due to data- and configuration-related perturbations. In response to such data and configuration variances, the *robustness* property [i.e., minor modifications to the (future) inputs of DNNs must not alter its outputs] has been extensively studied in the DL community. Robustness is the most noteworthy correctness property of DNNs. Assuring robustness is critically important to prevent artificial intelligence systems from experiencing environmental perturbations.

Previous research on DNN robustness has often focused on single impacts, as noted in Pham et al.[5] For example, Shu et al.[6] proposed a new Omni solution with a multimodel ensemble to address the impact of well-crafted adversarial samples on identification performance. Pinto et al.[7] observed that Mixup can significantly reduce performance when detecting out-of-distribution (OOD) samples. Xiao et al.[4] investigated the impact of CPU multithreading on DNN systems, while Pham et al.[5] examined the performance of identical models with random seeds and nondeterminism-introducing factors under different training runs. Dong

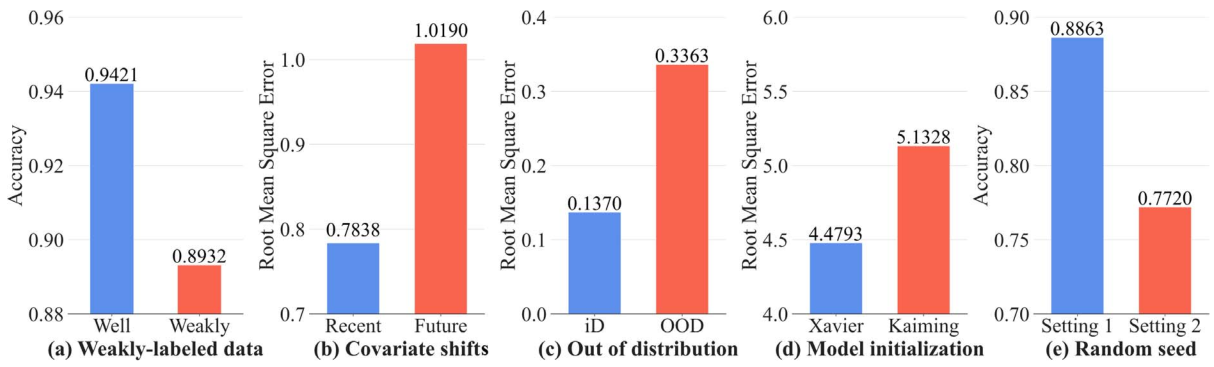

FIGURE 1. A range of applications in DNN projects, with their corresponding data and configurations. Accuracy on classification and root-mean-square error (RMSE) on regression tasks. (a) Weakly labeled data. (b) Covariate shifts. (c) Out of distribution (OOD). (d) Model initialization. (e) Random seed. SGD: stochastic gradient descent.

TABLE 1. A range of applications in DNN projects, with their corresponding data and configurations.

		Configurations	
Application use cases	**Data**	**Network architecture**	**Learning method**
(a) Recycling waste classification	TrashNet*	ResNet+Attention[b]	SGD
(b) Traffic forecasting	Traffic prediction dataset†	GRU[c]	SGD
(c) Water quality forecasting	Water quality data‡	Dual HeadSSIM¶	Adam
(d) Renewable energy prediction	Deep forecast§	DL-STF#	RMSprop
(e) Medical diagnosis	Physionet 2017 dataset‖	ResNet[1]	Adam

GRU: gated recurrent unit. *GitHub. Dataset of trash images. https://github.com/garythung/trashnet. †Kaggle. Traffic prediction dataset. https://www.kaggle.com/datasets/fedesoriano/traffic-prediction-dataset. ‡Kaggle. Real-time water quality data. https://www.kaggle.com/datasets/ivivan/real-time-water-quality-data. §Github. Dataset of wind speed. https://github.com/amirstar/Deep-Forecast. ‖The PhysioNet/Computing in Cardiology Challenge 2017. https://physionet.org/content/challenge-2017/1.0.0/. ¶Dual-head sequence-to-sequence imputation model (Dual HeadSSIM[3]) denotes the dual-head bidirectional GRU structure with cross-head attention. #DL-based spatiotemporal forecasting (DL-STF[2]) denotes the spatiotemporal recurrent neural network.

et al.[8] investigated the correlation among the test coverage, robustness, and metrics associated with attacks and defenses in DNNs. These recent approaches provide some insights into the various factors affecting DNN robustness.

Fewer studies have been conducted on a holistic understanding of the combined effects of diverse data and configuration factors. Initially, we present influencing factors for data and configuration variance in various real-world domains (see Table 1). Subsequently, we investigate the impact of combinatorial factors on DNN robustness in image classification. Our findings reveal that these factors introduce composition effects often overlooked in current research. Leveraging this insight, we formulate a robustness property considering perturbations to both data and configurations. Additionally, our ongoing research focuses on a novel search-based variance prediction method designed to expedite the identification of influential factors affecting DNNs.

PERFORMANCE VARIANCES—DATA AND CONFIGURATIONS

DNN models suffer from performance variances due to imperfect data or configurations, which can produce unreliable learning systems for a range of applications.

IMPACTS OF SINGLE FACTORS

Table 1 briefly describes five representative DNN projects related to the missions[a] from the Commonwealth Scientific and Industrial Research Organisation (CSIRO), and Figure 1 presents the corresponding experimental results from these projects regarding data and configuration perturbations:

[a]CSIRO's missions: Partner with us to tackle Australia's greatest challenges, https://www.csiro.au/en/about/challenges-missions

TABLE 2. Robustness-affecting factors.

Surface	Factor	Target
Data (F_D)	F_1 Adversarial attack F_2 OOD	Input $\mathbb{X}$
	F_3 Label-flipping attack F_4 Label noise injection	Output $\mathbb{Y}$
Configuration (F_C)	F_5 Weight modification F_6 Bias modification F_7 Conv layer modification F_8 FC layer modification F_9 Number of threadings F_{10} Pseudorandom seeds ...	Model f_θ ...

1) *Weakly labeled data*, which comprising partially labeled training samples, can impair the DNN model's accuracy. In the TrashNet dataset, some plastic and glass bottle images have incorrect labels. RecycleNet[b] experienced a 4.89% accuracy drop attributed to these weakly labeled training samples.
2) *Covariate shifts*, which demonstrate a distribution shift across time periods, adversely impact the model's performance. Figure 1(b) illustrates substantial root-mean-square error (RMSE) variance in two recurrent neural network models[c] trained on distinct temporal data segments (i.e., 2016.11.1–2017.2.28 and 2017.3.1–2017.6.30) for addressing traffic (supply-chain) congestion.
3) *OOD data*, which are introduced by the open set, cause untrustworthy estimation of the DNN model's performance. Figure 1(c) shows the increasing RMSE on the OOD testing set, which reveals the performance degradation of a gated recurrent unit model[3] on the water quality forecasting problem.
4) *Model initialization* arises from varying model initialization methods. In our wind speed forecasting project,[2] we experimented with different weight initialization methods using the same dataset and implementation. Figure 1(d) reveals a 0.65 RMSE difference between He et al.'s[9] and Glorot and Bengio's[10] initializations.
5) *Random seed* introduces nondeterministic factors during DNN training with stochastic algorithms. Figure 1(e) illustrates accuracy variations in DNN-based electrocardiogram recognition for cardiac arrhythmia diagnoses.[1] We ran 30 training jobs on ResNet with different random seeds (while keeping other settings constant), resulting in average performance differences exceeding 10%.

IMPACTS OF COMBINATORIAL FACTORS

From the results in Figure 1, different factors can cause substantial performance variances. To further inspect the impacts of combinatorial factors, we exercised and compared the eight executions of arbitrary combinations with three factors from Table 2 [i.e., an adversarial attack (F_1), a label-flipping attack (F_3), and weight modification (F_5)] to an image recognition model trained using the CIFAR-10 dataset. In Figure 2, the baseline model achieves 77.24% accuracy without perturbations, and we configure the settings regarding "F_1" by the fast gradient sign method[11] adversarial attack with $\sigma = 0.003$ [defined in (2)] "F_3" by multiclass label flipping[12] on 20% training data, and "F_5" by a random-weight parameter matrix modification. We could obtain two essential observations: 1) Combinatorial factors degrade the performance of DNN models, but the impacts cannot be inferred by the results on individual factors (e.g., "F_1" decreases the accuracy by 30.72% than baseline, "F_5" yields an accuracy drop of 34.75%, "$F_1 + F_5$" is not 65.47% but 57.79% instead); and 2) combinatorial factors may have some neutralized settings (e.g., "$F_1 + F_5$" are slightly more effective than "$F_1 + F_3 + F_5$" to degrade DNN accuracy).

[b]GitHub, sangminwoo/Recyclenet: Attentional Learning of Trash Classification. [Online]. Available: https://github.com/sangminwoo/RecycleNet
[c]Kaggle, Traffic Prediction: GRU. [Online]. Available: https://www.kaggle.com/code/karnikakapoor/traffic-prediction-gru

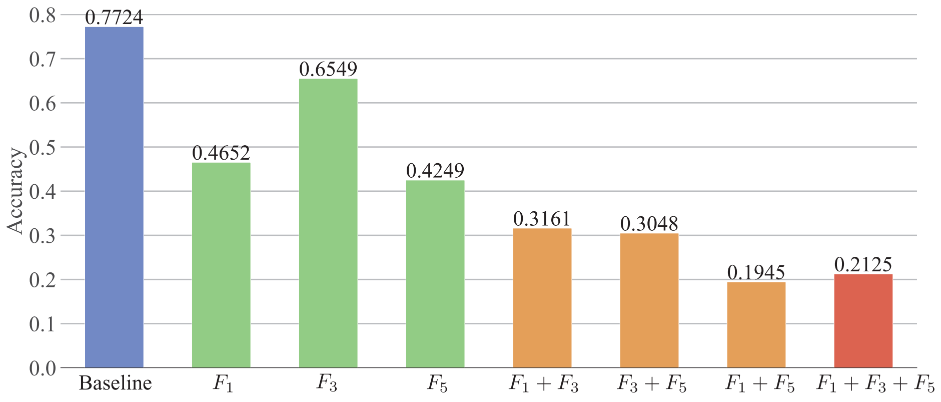

FIGURE 2. Impacts of combinatorial factors on an image classification DNN model.

ROBUSTNESS REGARDING DATA AND CONFIGURATIONS

The DNN model can be represented as a function $f_\theta : \mathbb{X} \to \mathbb{Y}$, which accepts an input $x \in \mathbb{X} \subseteq \mathbb{R}^n$ and returns an output $y \in \mathbb{Y} \subseteq \mathbb{R}^m$, where $\theta \in \Theta$ denotes the model parameter/architecture configuration, and $\mathbb{X}$ and $\mathbb{Y}$ are the inputs and outputs in the real number domain with n and m dimensions, respectively. Formally, we can train a DNN model by

$$f_\theta = \mathrm{train}(\mathbb{X}, \mathbb{Y}, \theta). \tag{1}$$

From this investigation, many factors exert single or combinatorial effects on the robustness performance of DL models in practical scenarios, such as recycling waste classification and traffic forecasting. Our previous investigation identified common robustness factors on two DL surfaces: data and configurations. These factors are summarized in Table 2. A robust DNN should handle small perturbations in both data and configuration, meaning that it should 1) always produce the same output for a slightly perturbed input, 2) produce correct results in the presence of noisy labels, and 3) maintain stable accuracy despite perturbed configurations. Equations (2)–(4) define these three aspects of DNN robustness with respect to data and configurations.

- *Data*: For supervised learning, a DNN model predicts the label of a sample through capturing the pattern between training data and their predefined labels. A trained DNN model f_θ is treated as robust if the input–output relation (x, y) always holds under perturbed inputs and outputs.
  - *Perturbed inputs*: During the training period, perturbed inputs are commonly imposed and can mislead the learning process. In response to the perturbations, a robust DNN can be formalized as

$$\begin{aligned}&\forall x \in \mathbb{X}, \hat{x} \in \mathbb{X},\\ &\|x - \hat{x}\|_{\mathrm{p}} < \sigma \Rightarrow f_\theta(\hat{x}) = y \in \mathbb{Y}\end{aligned} \tag{2}$$

  where $\hat{x}$ denotes the perturbed inputs under p normalization with σ distance (the degree of perturbations) to the original input x as the trained robust model can accept a perturbed input and return the consistent output. Adversarial attack (F_1) is regarded as the different forms of $\hat{x}$.
  - *Perturbed outputs*: If the training dataset contains corrupted or noisy output $(\hat{y} \in \hat{\mathbb{Y}})$ (e.g., under a δ distance to the ground-truth output y), a robust model f_θ can still maintain the correct prediction.

$$\begin{aligned}&\forall (x, y) \in (\mathbb{X}, \mathbb{Y}), \hat{y} = y \times \tau,\\ &\|f_\theta(x) - \hat{y}\|_{\mathrm{p}} < \delta \Rightarrow f_\theta(x) = y,\\ &\text{subject to } f_\theta = \mathrm{train}(\mathbb{X}, \hat{\mathbb{Y}}, \theta)\end{aligned} \tag{3}$$

  where $\tau \in \mathbb{R}^{m \times m}$ denotes the label transition probability matrix.
- *Configurations*: In addition to data factors, the training variance from configurations (e.g., model initialization and hyperparameters) is another reason for the robustness issues of DNNs. A trained DNN is robust if prediction variance is less susceptible to configuration perturbations.

$$\begin{aligned}&\forall x \in \mathbb{X}, \theta \in \Theta, \hat{\theta} \in \Theta,\\ &\|\theta - \hat{\theta}\|_{\mathrm{p}} < \eta \Rightarrow f_\theta(x) = f_{\hat{\theta}}(x) = y \in \mathbb{Y}\end{aligned} \tag{4}$$

where $\hat{\theta}$ denotes the configuration under p normalization with η distance (configuration differences) to the configuration θ.

Following these definitions, we can measure DNN robustness by the *cumulative confidence decision boundary* (*C-CDD*) through estimating the confidence of a model f_θ to predict all the samples from $\mathbb{X}$. Equation (5) presents a relative distance from the decision boundary to the human-desired classes.

$$\mathrm{C-CDD}(\mathbb{X}, f_\theta) = \mathbb{E}_{x\in\mathbb{X}}(f_\theta(x)[i] - f_\theta(x)[j]) \tag{5}$$

where $\mathbb{E}$ denotes the expectation operation, i denotes the human-desired class, and j is the predicted class with the maximum prediction probability. A lower C-CDD score indicates that the model potentially has higher uncertainties in its predictions.

SEARCH-BASED VARIANCE PREDICTION

Given the two perturbation surfaces [i.e., data (F_D) and configuration (F_C) in Table 2], we can obtain a perturbation set T_{F_i} with different predefined values for factor $F_i \in \{F_1, \ldots, F_N\}$. Perturbation strategy $ps \in T_{F_1} \ldots \times T_{F_i} \ldots \times T_{F_N}$ is a perturbation instance from all modifications given all factor combinations. Hence, modified inputs, outputs, and configurations from the strategy ps can be obtained by

$$\hat{\mathbb{X}}, \hat{\mathbb{Y}}, \hat{\theta} = \mathrm{perturb}_{\mathrm{ps}}(\mathbb{X}, \mathbb{Y}, \theta). \tag{6}$$

Given N factors with each having T modifications to the data and (or) configurations, we will have combinations of 2^{T*N} perturbation strategies to evaluate the robustness of a DNN model. Inspired by search-based software engineering, we present a search-based framework to conduct *selective optimization* and reduce the searching space, as shown in Figure 3. The perturbation pool collects all perturbation strategies, and we will select several strategies as the initialization settings for the investigation module. Here, the selected strategies are utilized to generate perturbed data samples and configuration settings via (6). The investigation module contains three main procedures, i.e., select, evaluate, and transform (optional), which can be leveraged for multiple solutions to the combinatorial optimization problem. 1) Select: the framework selects the candidate strategies from the perturbation pool. 2) Evaluate: the performance variances from the selected strategies are evaluated in this step. 3) Transform (optional): after the evaluation, some strategies with satisfactory results may be further transformed to jump out of the local search space. Here, the transformed strategies and the original satisfactory strategies are mixed for the next round of selective optimization. For this search-based investigation framework, the optimal $\mathrm{ps}^\star$ refers to finding the largest performance variances $\mathrm{PV}(\mathrm{ps})$

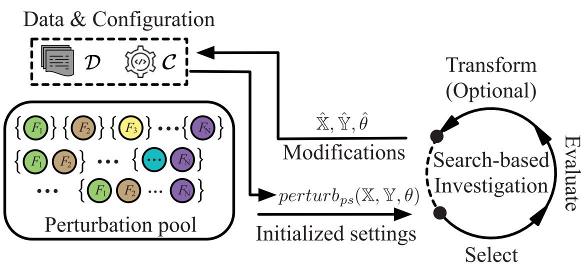
Data & Configuration
$\mathcal{D}$
$\mathcal{C}$
$\{F_1\}\{F_2\}\{F_3\}\cdots\{F_K\}$
$\{F_1 F_2\}\cdots\{F_3 F_K\}$
$\cdots\{F_1 F_2\cdots F_K\}$
Perturbation pool
$\hat{\mathbb{X}}, \hat{\mathbb{Y}}, \hat{\theta}$
Modifications
$perturb_{ps}(\mathbb{X}, \mathbb{Y}, \theta)$
Initialized settings
Transform (Optional)
Search-based Investigation
Evaluate
Select

FIGURE 3. Search-based investigation framework.

$$\begin{aligned} \mathrm{ps}^\star &= \max_{\mathrm{ps}} \mathrm{PV}(\mathrm{ps}) \\ &= \max_{\mathrm{ps}} |\mathrm{C-CDD}(\hat{\mathbb{X}}, f_{\hat{\theta}}) - \mathrm{C-CDD}(\mathbb{X}, f_\theta)| \end{aligned} \tag{7}$$

where $\hat{\mathbb{X}}$ and $\hat{\theta}$ are derived from (6).

Our proposed framework is highly extensible and can accommodate mainstream search-based approaches by fitting the objective function in (7). The representative mainstream approaches, including the evolutionary algorithm (EA),[13] reinforcement learning (RL),[14] and sequential model-based optimization (SMBO),[15] can share the same steps but may occur in different orders.

EA

The EA is a fast global optimization algorithm inspired by biological terminology, whereas an initial set of individual solutions are introduced to small, random changes and updated iteratively. Here, we use the EA to search from the perturbation pool for the expected $\mathrm{ps}^\star$. The EA performs the iterative way in the investigation module, and for $\forall \mathrm{ps}_i^{g+1}$ in the $(g+1)$-th generation, the transform step is divided as mutate and crossover:

- *Select*. First, three strategies, $\mathrm{ps}_{r_1}^g$, $\mathrm{ps}_{r_2}^g$, and $\mathrm{ps}_{r_3}^g$, are randomly selected from the candidate strategies in the g-th generation, where the subscript of the strategy (r) indicates the index of the candidate in the current generation. The indices of these three strategies should satisfy $r_1 \neq r_2 \neq r_3 \neq i$, where i is the index of the expected strategy ps_i^{g+1}.
- *Mutate*: A candidate ps_i^{g+1} can be obtained by the mutation operation $\mathrm{ps}_i^{g+1} = \mathrm{ps}_{r_1}^g + \epsilon(\mathrm{ps}_{r_2}^g - \mathrm{ps}_{r_3}^g)$, where $\epsilon \in (0,1]$ denotes a scaling coefficient for the mutation proportion.
- *Crossover*: The strategy ps can be treated as a K-length vector that contains a sequence of modification methods. Let $\mathrm{ps}_i^g[j]$ denote the j-th $(1 \leq j \leq K)$ modification in the perturbation strategy. The crossover follows a random probability value from the uniform distribution, i.e.,

$r_i^g[j] \sim U(0,1)$. The $(g+1)$-th generation of each particular modification can be obtained as

$$\mathrm{ps}_i^{g+1}[j] = \begin{cases} \mathrm{ps}_i^g[j], & \text{if } r_i^{g+1}[j] < p_r \\ \mathrm{ps}_i^{g+1}[j], & \text{otherwise} \end{cases} \tag{8}$$

where p_r is a user-specified replacement rate (e.g., 0.9).

- *Evaluate*: For each newly generated strategy ps_i^{g+1}, we produce a different DNN model by (1) and return the strategy with a lower C-CDD score from $\{\mathrm{ps}_i^g, \mathrm{ps}_i^{g+1}\}$.

The EA can balance the cost and satisfaction from the user's specifications and engineering requirements by configuring the value of ϵ and p_r to control how drastically it reaches a fitness degree constrained by (7).

RL

The combinatorial optimization problem can be solved using a sequential decision-making process called *RL*. The RL agent learns to make decisions by receiving positive or negative rewards during state transitions, which represent the model's robustness with parameter $\hat{\theta}$. The agent improves over time by gaining more experience. The action selection is based on the reward R from the robustness variance. This process can be represented as a Markov decision process (MDP) $\mathrm{MDP} = \langle S, A, R, T, \gamma \rangle$. Here, we elaborate on each of the tuple elements:

- *State* (S) : We represent the network state $s \in S$ through a pair of features, $s = \langle f_{\hat{\theta}}, \mathrm{C-CDD} \rangle$, for the newly produced DNN model based on (6) and the corresponding cumulative confidence score.
- *Action* (A) : During the learning process, the configurable settings from the strategy are selected from the perturbation pool, whereas $a \in A$ denotes the changing settings.
- *Transition* (T) : $T_a(s, s') = T_a(s_{t+1} = s' \mid s_t = s)$ denotes the probability of the transition from s to s' under action a at time t (the t-th selection of the strategy).
- *Reward* (R) : RL acts in an MDP to find a policy $A \times S \to [0,1]$ that maximizes the expected cumulative reward, and $r \in R$ denotes the immediate rewards from t to $t+1$ transition time.
- *Scaling rate* (γ) : A lower γ value contributes more weight to the short-term reward, while a higher γ value reflects that learning actions would reward more weight toward long-term ones.

With the aforementioned instances, a set of strategies are first selected and evaluated by the C-CDD. The agent receives and learns the numerical rewards from the transitions, which can be formed as a sequence, i.e., $s_0, a_0, r_1, s_1, a_1, r_2, \cdots$. Then, the sequence can be utilized in the learning iteration until convergence, reaching our defined objective.

SMBO

SMBO is another alternative option for addressing the problems regarding searching for configurable perturbation strategies. Our framework is interchangeable to fit with the components and facets that approximate the optimal solution based on a sequential model. SMBO utilizes previous belief distribution to fit a Gaussian mixture model for the desired strategy to the DNN model f_θ. Specifically, for the search-based variance prediction, SMBO conducts two steps for estimation of the global optimal $\mathrm{ps}^\star$ with the largest performance variance:

1) *Evaluate*: SMBO incrementally accepts from zero to t number of distinct perturbation strategies and constructs an acquisition function using the posterior for f_θ with the previous $t-1$ evaluations $\{(\mathrm{ps}_i, \mathrm{PV}(\mathrm{ps}_i)\}_{i=1}^{t-1}$.
2) *Select*: The acquisition function selects the next potential perturbation strategy ps_{i+1} and calculates the corresponding $\mathrm{PV}(\mathrm{ps}_{i+1})$ for the update of the acquisition function. The acquisition function determines the next points (another strategy) worth being evaluated, which keeps the balance between different perturbation factors (in the sense of which factor is more essential) and exploration of the strategies (in the sense of which settings are more effective to degrade).

The fitness agreement is still determined by humans; when SMBO reaches a given value or a specific threshold time (i.e., the total number of evaluation times t), the potential strategy from the current acquisition function is regarded as approximation of the optimal settings.

CONCLUSION

In this article, we presented and studied the two primary sources of performance variances in DL, i.e., imperfect data and configurations. We showcased the performance variances in real-world applications in various CSIRO missions. The proposed framework sheds light on future research in robust DL by providing a promising groundwork for predicting influential factors with search-based optimizations. Furthermore, exploring the potential solutions for mitigating the combinatorial effects from multiple robustness-affecting issues is promising. In future work, we incorporate iterative optimization into the related countermeasures, optimizing each subproblem iteratively.

REFERENCES

1. R. Wang, J. Fan, and Y. Li, “Deep multi-scale fusion neural network for multi-class arrhythmia detection,” *IEEE J. Biomed. Health Inform.*, vol. 24, no. 9, pp. 2461–2472, Sep. 2020, doi: 10.1109/JBHI.2020.2981526.
2. A. Ghaderi, B. M. Sanandaji, and F. Ghaderi, “Deep forecast: Deep learning-based spatio-temporal forecasting,” in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, *arXiv: https://arxiv.org/abs/1707.08110*.
3. Y. Zhang and P. J. Thorburn, “A dual-head attention model for time series data imputation,” *Comput. Electron. Agriculture*, vol. 189, Oct. 2021, Art. no. 106377, doi: 10.1016/j.compag.2021.106377.
4. G. Xiao, J. Liu, Z. Zheng, and Y. Sui, “Nondeterministic impact of CPU multithreading on training deep learning systems,” in *Proc. IEEE 32nd Int. Symp. Softw. Rel. Eng.*, 2021, pp. 557–568.
5. H. V. Pham et al., “Problems and opportunities in training deep learning software systems: An analysis of variance,” in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2020, pp. 771–783, doi: 10.1145/3324884.3416545.
6. R. Shu, T. Xia, L. Williams, and T. Menzies, “Omni: Automated ensemble with unexpected models against adversarial evasion attack,” *Empirical Softw. Eng.*, vol. 27, 2022, Art. no. 26, doi: 10.1007/s10664-021-10064-8.
7. F. Pinto, H. Yang, S. N. Lim, P. Torr, and P. Dokania, “Using mixup as a regularizer can surprisingly improve accuracy and out-of-distribution robustness,” in *Proc. 35th Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2022, pp. 14,608–14,622.
8. Y. Dong et al., “An empirical study on correlation between coverage and robustness for deep neural networks,” in *Proc. 25th Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, 2020, pp. 73–82, doi: 10.1109/ICECCS51672.2020.00016.
9. K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 1026–1034.
10. X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proc. 13th Int. Conf. Artif. Intell. Statist. JMLR Workshop Conf.*, 2010, pp. 249–256.
11. I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.
12. Z. Zhang and M. Sabuncu, “Generalized cross entropy loss for training deep neural networks with noisy labels,” in *Proc. 31st Adv. Neural Inf. Process. Syst.*, 2018, *arXiv:1805.07836*.
13. N. Awad, N. Mallik, and F. Hutter, “DEHB: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization,” in *Proc. 29th Int. Joint Conf. Artif. Intell.*, 2021, pp. 2147–2153.
14. N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, “Reinforcement learning for combinatorial optimization: A survey,” *Comput. Oper. Res.*, vol. 134, Oct. 2021, Art. no. 105400, doi: 10.1016/j.cor.2021.105400.
15. A. Candelieri and F. Archetti, “Sequential model based optimization with black-box constraints: Feasibility determination via machine learning,” *AIP Conf. Proc.*, vol. 2070, no. 1, Feb. 2019, Art. no. 020010, doi: 10.1063/1.5089977.

GUANQIN ZHANG is a Ph.D. candidate in software engineering and computer science at the University of New South Wales, Sydney, 2052, Australia. His research interests include artificial intelligence, software engineering, and program verification. Zhang received his M.Res. degree from the University of Technology Sydney. Contact him at guanqin.zhang@unsw.edu.au.

JIANKUN SUN is with the University of Science and Technology Beijing, Beijing, 100083, China. His research interests include software engineering, artificial intelligence, and software security. Sun received his Ph.D. degree in computer science from the University of Science and Technology Beijing, Beijing, China. Contact him at sunjk@xs.ustb.edu.cn.

FENG XU is a Ph.D. candidate in software engineering and computer science at the University of New South Wales, Sydney, 2052, Australia. His research interests include software engineering, program analysis, and artificial intelligence. Xu received his master’s degree from the University of Sydney. Contact him at fengxxu@outlook.com.

YULEI SUI is a scientia associate professor at the School of Computer Science and Engineering, the University of New South Wales, Sydney, 2052, Australia. His research interests broadly include program analysis, software engineering, and security. Sui received his Ph.D. degree in software engineering and computer science from the University of New South Wales. He is a Senior Member of IEEE. Contact him at yulei.sui@unsw.edu.au.

H.M.N. DILUM BANDARA is a senior research scientist with the Architecture and Analytics Platforms Team, Data61, the Commonwealth Scientific and Industrial Research Organisation, Sydney, 2015, Australia. He is also a conjoint senior lecturer at the University of New South Wales. His research interests include distributed systems, computer security, and software architecture. Bandara received his Ph.D. degree in electrical and computer engineering from the Colorado State University, Fort Collins, USA. He is a Senior Member of IEEE and a chartered engineer at the Institution of Engineers Sri Lanka. Contact him at dilum.bandara@data61.csiro.au.

SHIPING CHEN is a senior principal research scientist with Data61, the Commonwealth Scientific and Industrial Research Organisation, Sydney, 2015, Australia. He is also a conjoint professor at the University of New South Wales, the University of Sydney, and Macquarie University. His research interests include application security, blockchain, and digital services. Chen received his Ph.D. degree from the University of New South Wales. He is a Senior Member of IEEE and a fellow of the Institute of Engineering Technology. Contact him at shiping.chen@data61.csiro.au.

TIM MENZIES is a professor of computer science at North Carolina State University, Raleigh, NC, 27695, USA. His research interests include software engineering, data mining, and artificial intelligence. Menzies received his Ph.D. degree in computer science from the University of New South Wales. He is a Fellow of IEEE. Contact him at timm@ieee.org.