Towards Building Accountable Deep Neural Model by Interpreting Classification Accuracy

Rangeet Pan, Giang Nguyen, and Shibbir Ahmed

Iowa State University, Ames, IA, 50011, USA,
{rangeet,gnguyen,shibbir}@iastate.edu

Abstract. Deep neural networks have gained large popularity in the past decade. Compared to the non-ML based system, there is not much existing verification mechanism for these learning-based models. One such verification technique includes the contract between the advertised trustworthiness and the actual performance of these models. Our study identifies that due to the random initialization, accuracy i.e., trustworthiness changes even if the experimental setup remains unchanged. In order to address this issue, we have proposed an approach to identify these randomly initialized parameters and apply the search algorithm to find the corresponding value that provides the optimum accuracy. We have also proposed a specification language to restrict the learning process that helps a DNN model to achieve accountability in the aspect of classification accuracy.
Keywords: Accountability, Deep Neural Network, Interpretability , Model Verification.

1 Introduction

With the increasing popularity of Machine Learning (ML) based systems, the verification and validation of such systems are necessary. Although there is a vast amount of research carried out on the non-ML validation framework, there is a few validation based research done on the ML-based systems due to the nature of probabilistic and complexity. Without a proper validation framework, one might ask about the way to hold ML-based systems being accountable to the expectation. For instance, the contract made with the end-user by exposing the trustworthiness of these systems in terms of accuracy. But this metric of trustworthiness is variant even if the whole experimental setup remains the same. In this study, we address such problems and have proposed a programming language infrastructure to measure the optimum accuracy.

The recent works on this field can be primarily categorized into two sections, verifying a model to be accountable for the assigned task [1,2,3,4,5] and holding the accountability by making ML models more robust [6,7,8]. The prior works have focused on validating the input influence [9], explaining the models to make the black-box system grayer. However, these systems either hold domain knowledge or model operation knowledge as the key to increase the explainability. In this study, we have combined these two type of knowledge and propose a system that can takes the input dataset, model operations and their distributions as input and produce a optimum accuracy rather than a single value that changes everytime an ML model has been trained with same dataset and same experimental setup. We leverage the information to propose a specification language *ADNN* that verify the model's capability.

Problem Statement. Our proposed work is focused on the image-based deep neural network (DNN) classifier. We have identified multiple pieces of evidence that with the same experimental setup, one DNN model can produce different output evaluation metrics e.g., accuracy. In the learning process, these models begin the operation with random initialization of several parameters e.g., seed, weight, bias. With this randomly initialized value, the learning process continues that ends up producing different results for a different execution. In order to address such a problem, we are proposing a mining-based distribution learning and specification language-based approach that helps the user to hold a model structure accountable for the output.

Our contribution to this study has been the following:

- We have mined the *Keras* documentation to identify the randomly initialized parameters.
- We have proposed a adaptive simulated annealing approach that produces the optimum value of accuracy.
- We have also proposed a specification langugage that validates whether a model can achieve a user defined accuracy or not.

2 Motivation

Everytimes we train ML models on the same dataset, we can achieve different accuracies. Taking figure 1 as an example, a user obtains various accuracy value when she train her model on one dataset. [1].

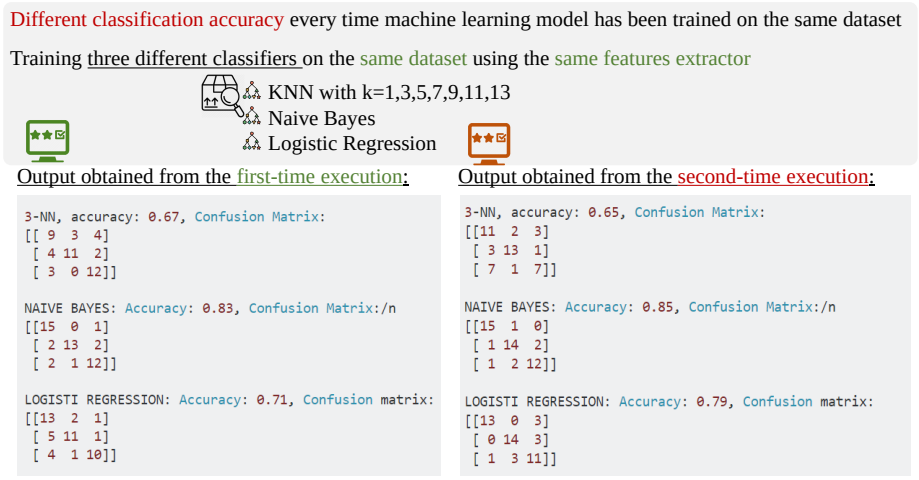
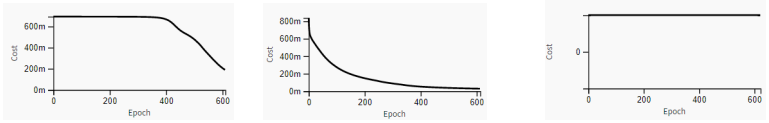


Fig. 1: Example of different accuracy on each execution using same experimental setup

[1] https://stackoverflow.com/questions/55775450/

In this scenario, different classification accuracy has been obtained with every execution of a machine learning model that has been trained on the same dataset. Three different classifiers e.g., k-nearest neighbors, Naive Bayes, and Logistic Regression have been compared using the accuracy of each model. With every execution, these models have been trained with the same experimental setup, different values of accuracy for each model have been obtained as illustrated in Figure 1. These results in a lack of trustworthiness of an ML-based model. Currently, we still do not have an appropriate answer for the results produced by ML models. However, we do know that initial bias and weight have contribution to the final results of a ML model. Moreover, one of the factors which cause the different accuracies of a ML models on the same dataset is different initial bias and weight. The obvious question would be can we increase the accountability of a ML model by obtaining the best accuracy among the possible accuracies produced by this model? He *et al.*[10] also mention that a ML model can be obstructed by bad initializations of weight and bias.



(a) Too small initialization (b) Appropriate initialization (c) Too large initialization

Fig. 2: Performance of a ML model on a dataset with different initializations

Figure 2 show the performance of a ML model on a dataset with different initializations. Particularly, if we select too small initialization, the model will take a long time to converge, whereas if we choose too large initialization, the model will never converge. Therefore, selecting an appropriate initialization is crucial for the performance of a ML model. In this project, we propose a method to make ML model more accountable by select appropriate initializations of weight and bias.

3 Related Work

3.1 Study on accountable ML model.

Validating Accountability. There is a vast amount of research on validating and holding an ML model accountable. One of the earliest validation technique proposed by Pulina and Tacchella [1] that utilizes abstraction to explain special types of a neural network, multi-layer perceptrons (MLP). [1] proposed a refinement approach to study the safety of MLP and repair them. However, this work can only be utilized if a network has six neurons. Gehr et al. [2] has proposed an infrastructure AI^2 that converts a neural network with convolution and fully connected layers to address the safety and robustness of the ML models. This work has addressed some issues, e.g., the trade-off between

precision and scalability, presenting an abstract representation of robustness in convolutional operation. However, this infrastructure does not work for a complex model, and it suffers from scalability issues. Du et al. [3] have surveyed models and research papers and have found that though there is a vast majority of work in machine learning has been done on increasing the explainability of the models. This work describes the clear categorization and complete overview of prevalent techniques to increase the interpretability of machine learning models aiming to help the community to understand the capabilities and weaknesses of different accountable approaches better. This study concludes that the prior works do not answer some question developers have e.g., *"why Q, not R"* or in our case why the accuracy changes with the same experimental setup and how we can believe an ML model if it does not provide a concrete contract to the end-user. Another survey [4] has been conducted on similar research artifacts and have concluded the overall theme for HCI researchers to hold an ML model accountable.

Holding Accountability. Jia et al. [8] proposed a programming language to prune the neural network to explain and interpret the model behavior and find bugs in the graph-based operations. RELUVAL [6] proposed a symbolic interval analysis to provide a formal guarantee of a deep neural network-based model. This work proposed a technique to formalize the dependency information of a network while the network operations propagate. Similar to the RELUVAL, [7] built a framework based on SMT solver to verify neural networks. Another area of research includes increasing robustness through crafting attacks and creating a defense against adversarial attack [11] that posses a threat against an ML model. Other studies [12,13] produce a verification process to defend against such attacks.

3.2 Accuracy validation

Ribeiro et al. [14] focused on the trustworthiness validation on an ML model and proposed LIME, a modular and interpretable approach to explaining the predictions of any model in an accountable manner.

In another study [15], the authors proposed an explainable DNN model that validates the prediction outcome. This work demonstrated that the abstract representation of DNN-based models can diagnose and interpret the working mechanism of the prediction task.

Zhang et al. [16] proposed an alternate convolutional neural network (CNN) based models that holds more informations than the traditional ones which makes the CNN model more accountable to the classification accuracy even though in some cases, accuracy may declined. So, accuracy validation is crucial while making an interpretable CNN model.

3.3 Accountability

According to the study [17], accountability in decision making represents the explanation about the "ongoing strategy". From the §2, we have found that a single model structure can provide different decision-making capabilities due to the assertion of the probabilistic distribution in the initialization process. The prior work [18] has already

demostrated that assertion based specification language can achieve accountability by reasoning about programs that behave randomly. In our proposed approach, we learn the unknown distribution of the initialization parameter by manually mining the *Keras* documentation. Then, we find the optimum value for the randomly initalized parameters using a greedy search algorithm. This approach will help the end-users to hold a model accountable in terms of the contract made between the model structure and the predicted output.

4 Background

In this section, we have discussed the traditional operations carried out in validating the accountability of a DNN model.

Forward Propagation. In this stage, a DNN model learns the features from the input. This stage includes random initialization, activation functions, feedforward, and loss function. A deep neural network can be represented as a fully connected graph. This graph consists of nodes and edges with associated values. These values are randomly initialized to start the training process. This step is very important because a wrong initialization can hamper the learning process and take a longer iteration to reach optimality. Also, it is never a good idea to initialize them with zero because the gradient will be computed as zero in the consecutive steps, and this will end up not learning anything from the input. The activation function has an impact on computation. However, it does not change the probability distribution of the computation. The main purpose of the activation function is to convert the representation of the value corresponds to the nodes according to the need. On the other hand, the loss function computes the difference between the expected and the actual result.

Backward Propagation. This is similar to the forward propagation, but only the difference is the way of computation. In this stage, the network operations are computed backward, and the derivative of each term has been computed to find the gradient that helps the learning process to localize the error and fix in the direction of the gradient.

5 Approach

In this section, we have proposed a comprehensive approach to address the discussed problem. In the §4, we have discussed the forward and backward propagation. In the two-step learning process, random initialization plays a crucial role in model validation. In Figure 3, we have depicted the overview of the proposed approach. We begin our approach by mining the documentation of *Keras* DNN library and manually identify the parameters responsible for the randomly initialization. Thereupon, we compute the optimized value for those parameters using an adaptive simulated annealing algorithm. The aforementioned process produces a near optimum value for the output metric. At last, we build a secification language that verify whether a desired accuracy can be achieved by a structure of the DNN. Our methodology has been discussed as follows.

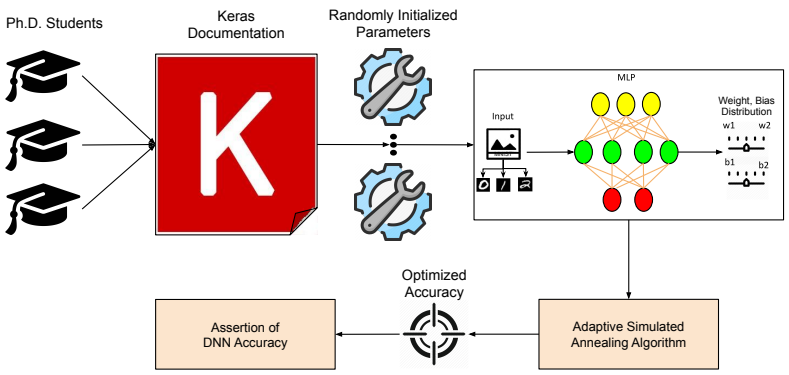


Fig. 3: Overview of the proposed approach

5.1 Optimized Loss Function

5.1.1 Identifying Randomly Initialized Parameter. In this phase, we have to understand the parameters that are subject to the random initialization. For this study, we have already mentioned that our focus has been on the deep learning-based image classifiers. To do so, we have mined the *Keras* [19] documentation to understand the implementation of DNN API. In this process, 3 Ph.D. students have individually gone through the implementation of all the functions. Our goal is two-fold here, 1) Identify the places, where random initialization has taken place that describe the learning-based system, 2) If we can mine the type of distribution from the implementation, then we can understand how the values correspond to that particular parameter has propagated to the output metrics. Our mining technique has identified a class of operation in *Keras* that is responsible for initializing the parameters, called *Initializers.*

```
model.add(Dense(64,
kernel_initializer='random_uniform',
bias_initializer='zeros'))
```

Listing 1.1: Example of initialization parameters in Keras

The above example of the *initializer* operation in *Keras* can initialize two parameters in the learning process, weight, and bias. The *kernel_initializer* is responsible for the weight value initialization, while *bias_initializer* is for bias value corresponds to that particular layer. This serves the purpose of finding the parameters that cause the value of the output to be variant with every execution.

To identify the distribution of these parameters, we have validated the implementation of the *initializer* class and have found that are a few different distributions that it supports and if not specifically given, which default distribution has been taken care of.

– Zeros: This initializes the parameter with 0. While, it may not be an issue for the bias, but if it has been initialized for weight, during the backpropagation, the deriva-

tive of the actual and computed value will be 1 for all cases that would make the learning process very hard to compensate the huge loss.

- Ones: This initializes the parameter with 1.
- Constant: Initializing with a constant is depended on the users' choice of the value.
- Normal Distribution: The parameter will be initialized with a normal distribution that takes input as mean (μ) and the standard distribution (σ) to describe the distribution.
- Uniform distribution: This initializes parameter with a uniform distribution where the range of a minimum and maximum value can be specified.
- Truncated Normal Distribution: The parameter is initialized with a truncated normal distribution where values greater than 2 standard deviations from mean are omitted. Here, the mean and standard deviation of the random values are used as arguments.
- LeCun uniform initializer: This initializes parameter using a uniform distribution where input unit numbers are specified as the limit for drawing samples from it.
- Glorot normal initializer: In this initializer, truncated normal distribution using the number of input and output units are used for drawing samples.

```
1  class Dense(Layer):
2  def __init__(self, units,
3  kernel_initializer='glorot_uniform',
4  bias_initializer='zeros',
5  ....
6  **kwargs):
```

Listing 1.2: Example of Keras's default setup

- Glorot uniform initializer: This initializes parameter using a uniform distribution where input unit numbers along with the output unit numbers are specified as the limit for drawing samples from it.
  We can observe how kernel and bias initializers are initialized from the following example,
- LeCun normal initializer: In this initializer, truncated normal distribution using the number of input units are used for drawing samples.

From the *Keras* documentation, we have found that weight and bias parameters are responsible for the change of the output metric in a DNN model. These two parameters are randomly initialized from the user-provided distribution or with the default one. Thus, our goal in this study is to find the value of the bias and weight s.t. the accuracy will be the highest or the loss value will be the least. This is a linear optimization problem, which is solvable in exponential time, but the nearest optimum solution can be achieved in polynomial time. Therefore, we have used an adaptive simulated annealing [20] to find the optimized value of the weight and bias combination that leads to better accuracy.

5.2 Accountable DNN Accuracy based on User Intent

In this approach, we have explained how an accountable DNN accuracy can be obtained based on user intent. In terms of user intent, we have introduced max_time, max_trial, max_gain which are described as follows:

- **max_time** In our approach, we have not restricted to a specific time rather we have provided the user the opportunity to give a certain maximum allowable time until the near-optimal accuracy can be obtained. To emphasize this approach we have empirically run the experiment with two different datasets varying δ and observe the accuracy based on user given max_time which has been explained in section 6.
- **max_trial** Similar to the max_time, we have not restricted the proposed approach to a specific trial as well. Instead, we have also allowed the user to set a certain maximum allowable trial until the near-optimal accuracy can be obtained independent of maximum allowable time. For clarifying the specific scenario of max_trial, we have conducted an experiment with MNIST, F-MNIST datasets varying δ and observe the accuracy based on the provided max_trial which has been explained in section 6.
- **max_gain** In addition to max_time and max_trial, we have used another user intent i.e., max_gain. The notion of max_gain is derived from the difference between actual accuracy and how much accuracy we obtained from our proposed approach. As, we have not confined our approach either to a specific time or a certain trial, we have also made this approach accountable by introducing the concept of user-defined maximum obtained gain in terms of accuracy of a DNN model. Moreover, in section 6we have shown experimental evaluation about how maximum gain can be achieved using three handmade models of each MNIST, F-MNIST datasets with varying δ in several trials.

In the Algorithm 1, we have utilized the users' intent and have discussed the methodology to perform the searching operation that could be beneficial to increase the accuracy. We have taken the value of the dataset, batch size, learning rate, and epoch parameters from the users as it varies from the choice of the problem and domain. As our proposed approach is currently able to support dense layers, we have taken the number of hidden dense layers present in the model architecture from the users. We have initialized the weight and bias from a random function and the size of the weight and bias is dependent on the size of the hidden layers that users provide. We have denoted the initial starting point of the weight and bias as $init\_weight$ and $init\_bias$ and there will be i number of different weight and bias present, where i denotes the number of allowable hidden layers needed to build the architecture of the model. Then, we have trained the model based on the architecture and the dataset provided. We have not shown the algorithm for the training as we have utilized the MLP implementation of the model building based on *Tensorflow* and in the *Tensorflow* model can be built by providing the weight and bias from the scratch and does not need additional coding to perform a model into its imperative one. Once we have trained the initial model, we have stored the loss and the accuracy of the starting model and have started the searching process. Our searching process is based on the user's intent of the previously defined three parameters i.e., max_time, max_trial, and max_gain. if one of the three conditions

Algorithm 1 ADNN

$dataset, batch\_size, learning\_rate, training\_epoch$=user's choice of problem
$count\_hidden = i$
$hidden\_size = [h_1, h_2, ..., h_i]$
$init\_weight = [random()_1, ..., random()_i]$
$init\_bias = [random()_1, ..., random()_i]$
$accuracy\_list = []$
$trial, gain, time\_diff = 0$
procedure $adnn\_train(max\_time, max\_trial, max\_gain)$
    $(x\_train, y\_train) = dataset.load()$
    $current\_time = now()$
    $\delta =$ Empirically chosen value(s)
    for each δ_k in δ **do**
        $loss, accuracy\_old = train(init\_weight, init\_bias,$
        $dataset, batch\_size, learning\_rate, training\_epoch)$
        while $trial \leq max\_trial$ & $gain \leq max\_gain$ & $time\_diff \leq max\_time$ **do**
            $init\_weight = init\_weight + \delta_k$
            $init\_bias = init\_bias + \delta_k$
            $loss, accuracy = train(init\_weight, init\_bias,$
            $dataset, batch\_size, learning\_rate, training\_epoch)$
            $accuracy\_list.append(accuracy - accuracy\_old)$
            $trial = trial + 1$
            $time\_diff = now - current\_time$
            $gain = max(accuracy\_list)$
    return $accuracy\_old + gain$

does not meet, we have stopped the searching process. During the searching, we have increased the weight and bias value based on the δ_k, where k can be drawn from a set of real numbers that have been chosen based on the empirical evaluation discussed in §6.2. We have trained the model with the new sets of weight and bias and store the new accuracy and loss values. We keep on doing the operations until any of the user-provided conditions are met. In the proposed approach, we have defined a metric to measure the increase of the accuracy in each that is defined as *gain*. We have defined *gain* as $gain = accuracy_T - accuracy_M$, where $accuracy_T$ and $accuracy_M$ represents as the accuracy of the model after a single trial and accuracy of the model at the start respectively. Once, the algorithm stops, our proposed approach reports the highest gain and the corresponding model to the users.

6 Evaluation

In this section, we discuss the result and the experiments computed to evaluate our proposed approach.

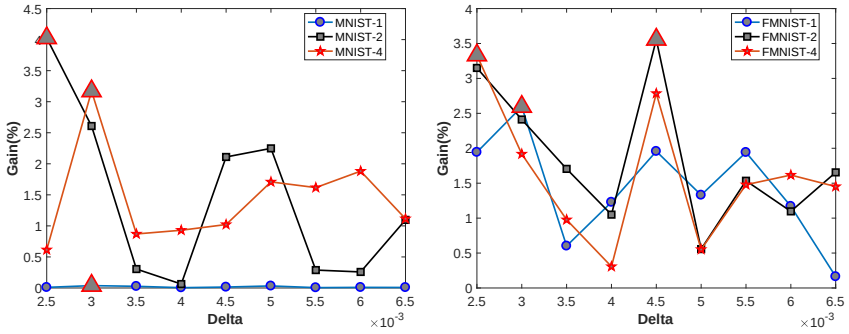
6.1 Experimental Setup

For evaluating our approach, we have chosen two different datasets

MNIST: This dataset [21] constitutes of different images of handwritten digits. It has 10 different sets of images, each representing a single digit. The training dataset contains 60,000 images and the testing dataset has 10,000 images. However, we have utilized the training dataset for the evaluation of our proposed approach. Furthermore, to observe the effect of our proposed approach, we have built 3 handmade models, 1) MNIST-1: This model has been built with one hidden layer with 256 neurons and the training accuracy of this model 88.6%, 2) MNIST-2: For this model, we have chosen two hidden layers with 256 neurons and the accuracy is 88.5%, and 3) MNIST-4: This model possesses 4 hidden layers with the same number of neurons and the accuracy is 87.2%.

Fashion MNIST: This dataset [22] is structurally equivalent to the MNIST but this dataset is built with different sets of clothes. Similar to the MNIST dataset, we have built 3 handmade models for the experimental purpose that we have denoted as FMNIST-1, FMNIST-2, and FMNIST-4, whose accuracies are 81.5%, 81.6%, and 80.2% respectively.

6.2 Results

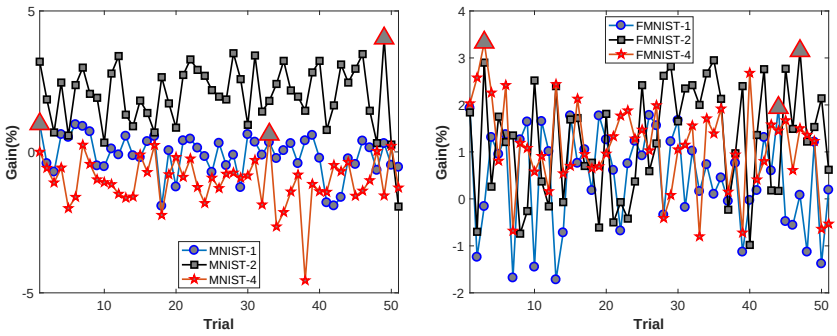


(a) MNIST (b) F-MNIST

Fig. 4: Experiment with varying δ and observe the gain of accuracy

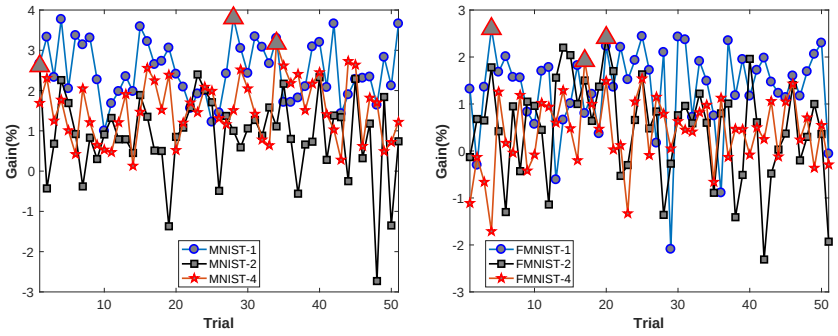
RQ1: What is the value of δ? In our approach to identifying the optimum solution for the weight and bias discussed in Algorithm 1, the step size δ has not been decided. Instead of varying the δ while finding the appropriate value for weight and bias, we empirically evaluated our approach with different values of δ and identified the most suitable value. In this experiment, we have performed the 50 iterations for each choice of δ and pick the best gain to report in Figure 4. Also, to perform this experiment, we have not restricted our algorithm with a specific time and any desired gain. In Figure 4, we varied the value of δ from 0.002 to 0.0065 and observed the gain of accuracy. We performed this experiment for both MNIST and F-MNIST for all three different types of the model discussed before. We have denoted the highest gain for each model

and dataset combination by an upward triangle. Our result suggests that five out of six evaluation conditions with δ=0.0025 and δ=0.003, the most gain has been achieved. For further evaluation, we have fixed the choice of the δ to either 0.0025 or 0.003.



(a) MNIST (b) F-MNIST

Fig. 5: Trial with δ=0.0025



(a) MNIST (b) F-MNIST

Fig. 6: Trial with δ=0.003

RQ2: How effective is this approach? To identify the gain achieved by our approach, we can utilize the same experimental setup discussed in the prior research question. We have found that our approach can increase the accuracy by 0.04%, 4.02%, and 3.17% for MNIST dataset with model MNIST-1, MNIST-2, and MNIST-4 respectively. Whereas, for the F-MNIST dataset, our maximum gain achieved is 2.56%, 3.55%, and 3.33% for FMNIST-1, FMNIST-2, and FMNIST-4 respectively. However, our approach takes

a significant amount of time to identify the solution as for each δ value we have iterated our solution findings approach for 50 iterations that is equivalent to training the model 50 times. We have performed another set of experiments with the δ value found in our previous experiments. The primary reason to perform these new sets of the experiment to identify a set of values for iteration for which our approach performs best that can be helpful for end-users to restrict their choices. In the Figure 5, we have performed the evaluation with the δ=0.0025 and similarly, in Figure 6, we have performed the experiments with δ=0.003. We have evaluated all 6 combinations of models with 2 datasets. For each combination of model and dataset, we have denoted the maximum gain of accuracy by an upward triangle in the plots. From the experimental evaluation, we have found that there is no particular trend that the algorithm followed that can be used to restrict the value of the trials. The maximum gain of accuracy is almost distributed all over the places and with this message, we have not restricted our algorithm with any prior choice of the value for a number of allowable trials.

RQ3: How efficient is this approach? We plan to measure the execution time for each iteration of evaluation and report the time needed to find the optimum solution for different δ and report the same.

7 Threat to Validity

8 Conclusion and Future Works

The author of this template is thankful to all those wonderful people who worked on this template.

References

1. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: International Conference on Computer Aided Verification, Springer (2010) 243–257
2. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP), IEEE (2018) 3–18
3. Du, M., Liu, N., Hu, X.: Techniques for interpretable machine learning. arXiv preprint arXiv:1808.00033 (2018)
4. Abdul, A., Vermeulen, J., Wang, D., Lim, B.Y., Kankanhalli, M.: Trends and trajectories for explainable, accountable and intelligible systems: An hci research agenda. In: Proceedings of the 2018 CHI conference on human factors in computing systems, ACM (2018) 582
5. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530 (2016)
6. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). (2018) 1599–1614

7. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification, Springer (2017) 97–117
8. Jia, Z., Padon, O., Thomas, J., Warszawski, T., Zaharia, M., Aiken, A.: Taso: Optimizing deep learning computation with automatic generation of graph substitutions. (2019)
9. Datta, A., Sen, S., Zick, Y.: Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In: 2016 IEEE symposium on security and privacy (SP), IEEE (2016) 598–617
10. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. (2015) 1026–1034
11. Papernot, N., McDaniel, P., Sinha, A., Wellman, M.: Towards the science of security and privacy in machine learning. arXiv preprint arXiv:1611.03814 (2016)
12. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM (2019) 731–744
13. Pan, R.: Static deep neural network analysis for robustness. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM (2019) 1238–1240
14. Ribeiro, M.T., Singh, S., Guestrin, C.: Why should i trust you?: Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, ACM (2016) 1135–1144
15. Du, M., Liu, N., Song, Q., Hu, X.: Towards explanation of dnn-based prediction with guided feature inversion. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM (2018) 1358–1367
16. Zhang, Q., Nian Wu, Y., Zhu, S.C.: Interpretable convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2018) 8827–8836
17. Veale, M., Van Kleek, M., Binns, R.: Fairness and accountability design needs for algorithmic support in high-stakes public sector decision-making. In: Proceedings of the 2018 chi conference on human factors in computing systems, ACM (2018) 440
18. Sampson, A., Panchekha, P., Mytkowicz, T., McKinley, K.S., Grossman, D., Ceze, L.: Expressing and verifying probabilistic assertions. In: ACM SIGPLAN Notices. Volume 49., ACM (2014) 112–122
19. Chollet, F.: Keras documentation. keras. io (2015)
20. Ingber, L.: Adaptive simulated annealing (asa): Lessons learned. arXiv preprint cs/0001018 (2000)
21. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (1998) 2278–2324
22. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)