

Distortion and Faults in Machine Learning Software^{*}

Shin NAKAJIMA

National Institute of Informatics
Tokyo, Japan

Abstract. Machine learning software, deep neural networks (DNN) software in particular, discerns valuable information from a large dataset, a set of data. Outcomes of such DNN programs are dependent on the quality of both learning programs and datasets. Unfortunately, the quality of datasets is difficult to be defined, because they are just samples. The quality assurance of DNN software is difficult, because resultant trained machine learning models are unknown prior to its development, and the validation is conducted indirectly in terms of prediction performance. This paper introduces a hypothesis that faults in the learning programs manifest themselves as distortions in trained machine learning models. Relative distortion degrees measured with appropriate observer functions may indicate that there are some hidden faults. The proposal is demonstrated with example cases of the MNIST dataset.

1 Introduction

Machine learning software, deep neural networks (DNN) software, is based on inductive methods to discern valuable information from a given vast amount of data [3]. The quality of such software is usually viewed from predication performance that obtained approximation functions exhibit for incoming data. Functional behavior of the resultant inference functions is dependent on trained learning models, which learning programs calculate with training datasets as their input.

The quality of DNN software is dependent on both the learning programs and training datasets; either or both is a source of degraded quality. The learning programs result in inappropriate trained learning models if they are not implemented faithfully in regard to well-designed machine learning algorithms [4]. Moreover, problematic datasets, suffering from sample selection bias [12] for example, have negative impacts on trained learning models.

Although the learning programs and datasets are sources to affect the quality of inference functions, they are more or less indirect. Trained learning models determine the quality directly, but have not been considered as *first-class citizens* to study quality issues. The models are important numeral data, but are intermediate in that they are synthesized by learning programs and then transferred to inference programs.

^{*} Presented at the 9th SOFL+MSVL Workshop in Shenzhen, November 2019.

This paper adapts a hypothesis that distortions in the trained learning models manifest themselves as faults resulting in quality degradation. Although such distortion is difficult to be measured directly as they are, relative distortion degrees can be defined. Moreover, this paper proposes a new way of generating datasets that show characteristics of the dataset diversity [9], which is supposed to be effective in testing machine learning software from various ways.

2 Machine Learning Software

2.1 Learning Programs

We consider supervised machine learning classifiers using networks of perceptrons [4] or deep neural networks [3]. Our goal is to synthesize, inductively from a large dataset, an approximation input-output relation classifying a multi-dimensional vector data \mathbf{a} into one of C categories. The given dataset LS is a set of N number of pairs, $\langle \mathbf{x}^n, t^n \rangle$ ($n = 1, \dots, N$), where a supervisor tag t^n takes a value of c ($c \in [1, \dots, C]$). A pair $\langle \mathbf{x}^n, t^n \rangle$ in LS means that \mathbf{x}^n is classified as t^n .

Given a learning model $y(W; \mathbf{x})$ as a multi-dimensional non-linear function, differentiable with respect to both learning parameters W and input data \mathbf{x} . Learning aims at obtaining a set of learning parameter values (W^*) by solving a numerical optimization problem.

$$W^* = \underset{W}{\operatorname{argmin}} \mathcal{E}(W; X), \quad \mathcal{E}(W; X) = \frac{1}{N} \sum_{n=1}^N \ell(y(W; \mathbf{x}^n), t^n)$$

The function $\ell(-, -)$ denotes distances between its two parameters, representing how much a calculated output $y(W; \mathbf{x}^n)$ differs from its accompanying supervisor tag value t^n .

We denote a function to calculate W^* as $\mathcal{L}_f(LS)$, which is a program to solve the above optimization problem with its input dataset LS . Moreover, we denote the empirical distribution of LS as ρ_{em} . W^* is a collection of learning parameter values to minimize the mean of ℓ under ρ_{em} .

From viewpoints of the software quality, $\mathcal{L}_f(LS)$, a learning program, is concerned with the product quality, because it must be a faithful implementation of a machine learning method, the supervised learning method for this case.

2.2 Inference Programs

We introduce another function $\mathcal{I}_f(\mathbf{x})$, using a trained learning model W^* or $y(W^*; \mathbf{x})$, calculates inference results of an incoming data \mathbf{x} .

For classification problems, the inference results are often expressed as probability that the data \mathbf{x} belongs to a category c . $Prob(W, \mathbf{a}; c)$, a function of c , is probability such that the data \mathbf{a} is classified as c . This $Prob$ is readily implemented in $\mathcal{I}_f(x)$ if we choose *Softmax* as an activation function of the output layer of the learning model.

The prediction performance of $\mathcal{I}_f(\mathbf{x})$ is, indeed, defined compactly as the accuracy of classification results for a dataset TS different from LS used to calculate W^* ; $TS = \{\langle \mathbf{x}^m, \mathbf{t}^m \rangle\}$. For a specified W , $Correct$ is a set-valued function to obtain a subset of data vectors in TS .

$$Correct(W; TS) \equiv \{ \mathbf{x}^m \mid c^* = \underset{c \in [1, \dots, C]}{\operatorname{argmax}} \operatorname{Prob}(W, \mathbf{x}^m; c) \wedge \mathbf{t}^m = c^* \}$$

If we express $|S|$ as a size of a set S , then an accuracy is defined as a ratio as below.

$$Accuracy(W; TS) = \frac{|Correct(W; TS)|}{|TS|}$$

Given a W^* obtained by $\mathcal{L}_f(LS)$, the predication performance of $\mathcal{I}_f(LS)$ is defined in terms of $Accuracy(W^*; TS)$ for a dataset TS different from LS . For an individual incoming data \mathbf{a} , a function of c $\operatorname{Prob}(W^*, \mathbf{a}; c)$ is a good performance measure.

2.3 Quality Issues

Loss and Accuracy An NN learning problem is non-convex optimization, and thus reaching a globally optimal solution is not guaranteed (e.g. [4]). The learning program $\mathcal{L}_f(LS)$ is iterated over epochs to search for solutions and is taken as converged when the value of the loss ($\mathcal{E}(W; LS)$) is not changed much between two consecutive epochs. The learning parameter values at this converged epoch are taken as W^* . The derived W^* may not be optimal, and thus an accuracy is calculated to ensure that the obtained W^* is appropriate. Moreover, W^* may be over-fitting to the training dataset LS .

In the course of the iteration, at an epoch e , the learning parameter values $W^{(e)}$ are extracted. The iteration is continued until the accuracy becomes satisfactory. Both $Accuracy(W^{(e)}, LS)$ and $Accuracy(W^{(e)}, TS)$ are monitored to ensure the training process goes as desired. If the accuracy of TS is not much different from the accuracy with LS , we may think that the learning result does not have the over-fitting problem.

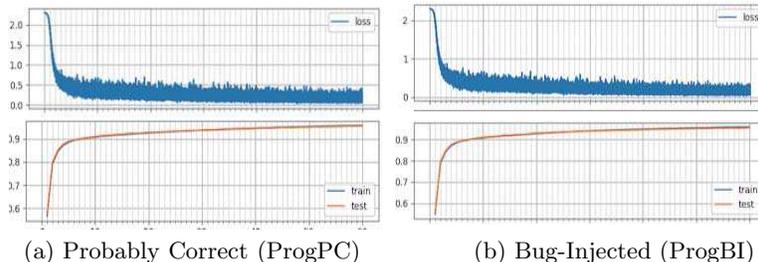


Fig. 1. Loss and Accuracy: MNIST Dataset

Figure 1 shows loss and accuracy graphs as epochs proceed measured during experiments¹. The graphs, for example in Figure 1 (a), actually demonstrate that the search converges at a certain desirable point in the solution space because the loss decreases to be almost stable below a certain threshold, and the accuracies of both TS and LS reach a satisfactory level of higher than 0.95. Figure 1 shows that the loss together with the accuracy may be good indicators to decide whether NN training processes behave well or not.

Sources of Faults Intuitively, NN machine learning software shows good quality if prediction performance of \mathcal{I}_f is acceptable. The graphs in Figure 1, however, depict that there is a counterexample case as discussed in [9], in which the learning task uses MNIST dataset, for classifying hand-written numbers.

Figure 1(a) are graphs of loss and accuracy of a probably correct implementation of NN learning program, while Figure 1(b) are those of a bug-injected program. The two graphs for loss are mostly the same to be converged. The two accuracy graphs are similar as well, although the program of Figure 1(b) has faults in it.

MNIST dataset is a standard benchmark and is supposed to be well-prepared free from any sample selection bias. A bug-injected program \mathcal{L}_f accepts a training dataset LS of MNIST and calculates a set of trained learning parameters W^* . Intuitively, this W^* is inappropriate, because a bug-injected program produces it. However, the accuracy graphs show that there is no clear sign of faults in the prediction results of \mathcal{I}_f , although its behavior is completely determined by the probably inappropriate W^* .

A question arises how faults in \mathcal{L}_f affect W^* , which follows another question whether such *faults* in W^* are observable.

3 Distortion Degrees

3.1 Observations and Hypotheses

Firstly, we introduce a few new concepts and notations. For two datasets DS_1 and DS_2 , a relation $DS_1 \preceq DS_2$ denotes that DS_2 is more distorted than DS_1 . For two sets of trained learning parameters W_1^* and W_2^* of the same machine learning model $y(W; _)$, a relation $W_1^* \preceq W_2^*$ denotes that W_2^* is more distorted than W_1^* . A question here is how to measure such degrees of the distortion. We assume a certain observer function obs and a relation \mathcal{R}_{obj} with a certain small threshold ϵ_{obs} such that $\mathcal{R}_{obj} = (|obs(W_1^*) - obs(W_2^*)| \leq \epsilon_{obs})$. The distortion relation is defined in terms of \mathcal{R}_{obj} , $W_1^* \preceq W_2^* \Leftrightarrow \neg \mathcal{R}_{obj}(W_1^*, W_2^*)$. We introduce below three hypotheses referring to these notions.

[Hyp-1] Given a training dataset LS , a machine learning program $\mathcal{L}_f(LS)$, either correct or faulty, derives its optimal solution W^* . For

¹ We return to the experiment later in Section 4.

the training dataset LS and a testing dataset TS , if both are appropriately sampled or both follows the same empirical distribution, then $Accuracy(W^*, TS)$ is almost the same as $Accuracy(W^*, LS)$.

[Hyp-2] For a training program \mathcal{L}_f and two training datasets LS_j ($j = 1$ or 2), if $W_j^* = \mathcal{L}_f(LS_j)$, then $LS_1 \preceq LS_2 \Rightarrow W_1^* \preceq W_2^*$.

[Hyp-3] For two training datasets LS_j ($j = 1$ or 2) such that $LS_1 \preceq LS_2$ and a certain appropriate obs , if \mathcal{L}_f is correct with respect to its functional specifications, then two results, $W_j^* = \mathcal{L}_f(LS_j)$, are almost the same, written as $W_1^* \approx W_2^*$ (or $obs(W_1^*) \approx obs(W_2^*)$). However, if \mathcal{L}'_f is a faulty implementation, then $W_1^* \prec W_2'^*$.

The accuracy graph in Figure 1 is an instance of [Hyp-1]. In Figure 1 (a), the accuracy graphs for TS and LS are mostly overlapped, and the same is true for the case of Figure 1 (b), which illustrates that the accuracy is satisfactory even if the learning programs is buggy.

Moreover, the example in [9] is an instance of the [Hyp-2] because of the followings. A training dataset $LS^{(K+1)}$ is obtained by adding a kind of disturbance signal to $LS^{(K)}$ so that $LS^{(K)} \preceq LS^{(K+1)}$. With an appropriate observer function obs_d , $\mathcal{R}_{obs_d}(W^{(K)*}, W^{(K+1)*})$ is falsified where $W^{(K)*} = \mathcal{L}_f(L^{(K)})$.

3.2 Generating Distorted Dataset

We propose a new test data generation method. We first explain the L-BFGS [14], which illustrates a simple way to calculate adversarial examples.

Given a dataset LS of $\{(\mathbf{x}^n, \mathbf{t}^n)\}$, $W^* = \mathcal{L}_f(LS)$. An adversarial example is a solution of an optimization problem;

$$\mathbf{x}^A = \underset{\mathbf{x}}{\operatorname{argmin}} A_\lambda(W^*; \mathbf{x}_S, t_T, \mathbf{x}),$$

$$A_\lambda(W^*; \mathbf{x}_S, t_T, \mathbf{x}) = \ell(y(W^*; \mathbf{x}), t_T) + \lambda \cdot \ell(\mathbf{x}, \mathbf{x}_S).$$

Such a data \mathbf{x}^A is visually close to a seed \mathbf{x}_S for human eyes, but is actually added a faint noise so as to induce miss-inference such that $y(W^*; \mathbf{x}^A) = t_T$.

Consider an optimization problem, in which a seed \mathbf{x}_S is \mathbf{x}^n and its target label t_T is t^n .

$$\mathbf{x}^{n*} = \underset{\mathbf{x}}{\operatorname{argmin}} A_\lambda(W^*; \mathbf{x}^n, t^n, \mathbf{x}).$$

The method is equivalent to constructing a new data \mathbf{x}^{n*} to be added small noises. Because the inferred label is not changed, \mathbf{x}^{n*} is not adversarial, but is *distorted* from the seed \mathbf{x}^n . When the value of the hyper-parameter λ is chosen to be very small, the distortion of \mathbf{x}^{n*} is large from \mathbf{x}^n . On the other hand, if λ is appropriate, the effects of the noises on \mathbf{x}^{n*} can be small so that the data is close to the original \mathbf{x}^n .

By applying the method to all the elements in LS , a new dataset is obtained to be $\{\langle \mathbf{x}^{n*}, \mathbf{t}^n \rangle\}$. We introduce a function T_A to generate such a dataset from LS and W^* .

$$T_A(\{\langle \mathbf{x}^n, \mathbf{t}^n \rangle\}, W^*) = \{ \langle \underset{\mathbf{x}}{\operatorname{argmin}} A_\lambda(W^*; \mathbf{x}^n, \mathbf{t}^n, \mathbf{x}), \mathbf{t}^n \rangle \}.$$

Now, $LS^{(K+1)} = T_A(LS^{(K)}, \mathcal{L}_f(LS^{(K)}))$ (for $K \geq 0$ and $LS^{(0)} = LS$).

3.3 Some Properties

This section presents some properties that generated datasets $LS^{(K)}$ satisfy; $LS^{(K)} = T_A(LS^{(K-1)}, \mathcal{L}_f(LS^{(K-1)}))$ where $LS^{(0)}$ is equal to be a given training dataset LS .

[Prop-1] $LS^{(K)}$ serves the same machine learning task as LS does.

We have that $W^{(0)*} = \mathcal{L}_f(LS)$. As the optimization problem with $W^{(0)*}$ indicates, $\mathbf{x}^{(n)*}$, an element of $LS^{(1)}$ does not deviate much from $\mathbf{x}^{(n)}$ in LS , and is almost the same as $\mathbf{x}^{(n)}$ in special cases. Therefore, $LS^{(1)}$ serves as the same machine learning task as LS does. Similarly, $LS^{(K)}$ serves as the same machine learning task as $LS^{(K-1)}$ does. By induction, $LS^{(K)}$ serves as the same machine learning task as LS does, although the deviation may be large. \square

[Prop-2] $LS^{(K)} \preceq LS^{(K+1)}$

The distortion relation is satisfied by construction if we take $LS^{(K)}$ as a starting criterion. \square

[Prop-3] $LS^{(K)}$ is more over-fitted to $W^{(K-1)*}$ than $LS^{(K-1)}$.

In the optimization problem, if the loss $\ell(\mathbf{y}(W^{(K-1)*}; \mathbf{x}^*), t_T)$ is small, \mathbf{x}^* in $LS^{(K)}$ can be considered to be well-fitted to $W^{(K-1)*}$ because the data reconstruct the supervisor tag t_T well. We make the loss is so small as in the above by choosing carefully an appropriate λ value. \square

[Prop-4] There exists a certain K_c such that, for all K to satisfy a relation $K \geq K_c$, $\operatorname{Accuracy}(W^{(K)*}, TS) \approx \operatorname{Accuracy}(W^{(K_c)*}, TS)$ and $\operatorname{Accuracy}(W^{(0)*}, TS) \geq \operatorname{Accuracy}(W^{(K_c)*}, TS)$. TS is a dataset different from LS , but follows the same empirical distribution ρ_{em} .

From Prop-3, we can see $LS^{(K)}$ is over-fitted to $W^{(K_c)*}$ if $K = K_c + 1$. Because $W^{(K)*} = \mathcal{L}_f(LS^{(K)})$ and both LS and TS follow the empirical distribution ρ_{em} , we have $\operatorname{Accuracy}(W^{(K)*}, TS) \approx \operatorname{Accuracy}(W^{(K_c)*}, TS)$. Furthermore, $LS \preceq LS^{(K_c)}$ implies $\operatorname{Accuracy}(W^{(0)*}, LS) \geq \operatorname{Accuracy}(W^{(K_c)*}, LS)$ and thus $\operatorname{Accuracy}(W^{(0)*}, TS) \geq \operatorname{Accuracy}(W^{(K_c)*}, TS)$. \square

[Prop-5] The dataset and trained learning model reach respectively LS^∞ and $W^{(\infty)*}$ if we repeatedly conduct the training \mathcal{L}_f and the dataset generation T_A interleavingly.

If we choose a K to be sufficiently larger than K_c , we have, from Prop-4, $Accuracy(W^{(K)*}, LS^{(K)}) \approx Accuracy(W^{(K_c)*}, LS^{(K)})$, which may imply that $Accuracy(W^{(K)*}, LS^{(K)}) \approx Accuracy(W^{(K+1)*}, LS^{(K+1)})$. From Prop-3, $LS^{(K+1)}$ is over-fitted to $W^{(K)*}$, and thus we have $LS^{(K)} \approx LS^{(K+1)}$, which implies that we can choose a representative $LS^{(\infty)}$ from them. Using this dataset, we have that $W^{(\infty)*} = \mathcal{L}_f(LS^{(\infty)})$, and that $W^{(\infty)*}$ is a representative. \square

4 A Case Study

4.1 MNIST Classification Problem

MNIST dataset is a standard problem of classifying handwritten numbers [6]. It consists of a training dataset LS of 60,000 vectors, and a testing dataset TS of 10,000. Both LS and TS are randomly selected from a pool of vectors, and thus are considered to follow the same empirical distribution. The machine learning task is to classify an input sheet, or a vector data, into one of ten categories from 0 to 9. A sheet is presented as 28×28 pixels, each taking a value between 0 and 255 to represent gray scales. Pixel values represent handwritten strokes, and a number appears as a specific pattern of these pixel values.

In the experiments, the learning model is a classical neural network with a hidden layer and an output layer. Activation function for neurons in the hidden layer is *ReLU*; its output is linear for positive input values and a constant zero for negatives. A *softmax* activation function is introduced so that the inference program \mathcal{I}_f returns probability that an incoming data belongs to the ten categories.

4.2 Experiments

We prepared two learning programs \mathcal{L}_f^{PC} and \mathcal{L}_f^{BI} . The former is a probably correct implementation of a learning algorithm, and the latter is a bug-injected version of the former. We conducted two experiments in parallel, one using \mathcal{L}_f^{PC} and the other with \mathcal{L}_f^{BI} , and made comparisons. Below, we use notations such as \mathcal{L}_f^{MD} where MD is either *PC* or *BI*.

Training with MNIST dataset We conducted trainings the MNIST training dataset $LS^{(0)}$; $W_{MD}^{(0)*} = \mathcal{L}_f^{MD}(LS^{(0)})$. Figure 1 illustrates several graphs to show their behavior, that are obtained in the training processes. Both accuracy graphs in Figure 1 show that $Accuracy(W_{MD}^{(0)*}, LS)$ and $Accuracy(W_{MD}^{(0)*}, TS)$ are mostly the same. In addition, $Accuracy(W_{PC}^{(0)*}, \cdot)$ and $Accuracy(W_{BI}^{(0)*}, \cdot)$ are indistinguishable. The above observation is consistent with **[Hyp-1]**.

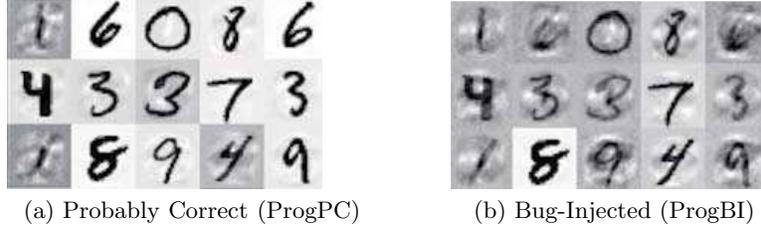


Fig. 2. Distorted Data

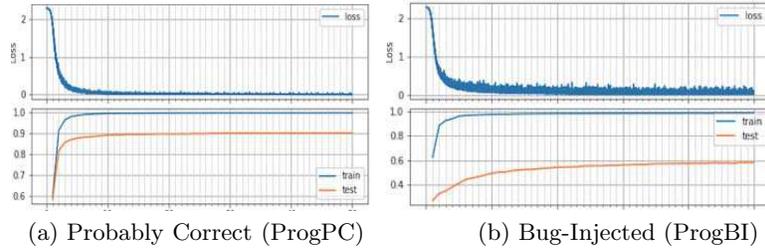


Fig. 3. Loss and Accuracy: Distorted Training Dataset

Generating Distorted Datasets We generated distorted datasets with the method described in Section 3.2. We introduce here short-hand notations such as $LS_{MD}^{(K)}$; $LS_{MD}^{(1)} = T_A(LS^{(0)}, \mathcal{L}_f^{MD}(LS^{(0)}))$.

Figure 2 shows a fragment of $LS_{MD}^{(1)}$. We recognize that all the data are not so clear as those of the original MNIST dataset and thus are considered distorted. We may consider them as $LS \preceq LS_{MD}^{(1)}$, which is an instance of **[Prop-2]**. Furthermore, for human eyes, Figure 2 (b) for the case with \mathcal{L}_f^{BI} is more distorted than Figure 2 (a) of \mathcal{L}_f^{PC} , which may be described as $LS_{PC}^{(1)} \preceq LS_{BI}^{(1)}$.

Training with Distorted Datasets We then conducted trainings the distorted dataset $LS_{MD}^{(1)}$; $W_{MD}^{(1)*} = \mathcal{L}_f^{MD}(LS_{MD}^{(1)})$. Figure 3 shows loss and accuracy graphs in their learning processes. Comparing Figure 3 with Figure 1 leads to the following observations.

Firstly, the overall loss values of Figure 3 are smaller than those of Figure 1 counterparts, and the metrics concerning with the differences ($\ell(-, -)$) are small for the distorted dataset cases.

Secondly, for the MNIST testing dataset TS , $Accuracy(W_{MD}^{(1)*}, TS)$ is lower than $Accuracy(W_{MD}^{(0)*}, TS)$, while $Accuracy(W_{MD}^{(1)*}, LS_{MD}^{(1)})$ reaches close to 100%. Together with the fact of $LS \preceq LS_{MD}^{(1)}$, the above implies $W_{MD}^{(0)*} \preceq W_{MD}^{(1)*}$, which is consistent with **[Hyp-2]**.

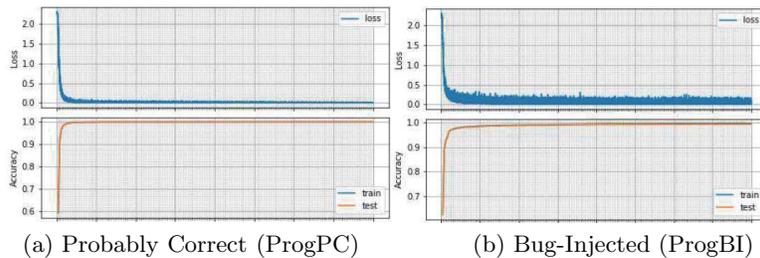


Fig. 4. Loss and Accuracy: Distorted Testing Dataset

Thirdly, we consider how much the accuracies differ. We define the relation \mathcal{R}_{obs} where $obs(W^*) = Accuracy(W^*, TS)$. Let \mathcal{R}_{obs}^{MD} be defined in terms of $Accuracy(W_{MD}^{(0)*}, TS)$ with a certain ϵ_{MD} . Comparing the graphs in Figure 1(a) and Figure 3(a), we observe, for \mathcal{L}_f^{PC} , ϵ_{PC} is about 0.1. Contrarily, for \mathcal{L}_f^{BI} from Figure 1(b) and Figure 3(b), ϵ_{BI} is about 0.4. If we choose a threshold to be about 0.2, the two cases are distinguishable.

Moreover, we define the \approx relation for ($\epsilon < 0.2$). As we know that \mathcal{L}_f^{PC} is probably correct and \mathcal{L}_f^{BI} is bug-injected, we have followings. (a) $W_{PC}^{(0)*} \approx W_{PC}^{(1)*}$, and (b) $W_{BI}^{(0)*} \prec W_{BI}^{(1)*}$. These are, indeed, consistent with **[Hyp-3]**.

Accuracy for Distorted Testing Datasets We generated distorted datasets from the MNIST testing dataset TS ; $TS_{MD}^{(1)} = T_A(TS, \mathcal{L}_f^{MD}(LS))$. We, then, checked the accuracy $Accuracy(W_{MD}^{(1)}, TS_{MD}^{(1)})$, whose monitored results are shown in Figure 4. $Accuracy(W_{MD}^{(1)}, TS_{MD}^{(1)})$ and $Accuracy(W_{MD}^{(1)}, LS_{MD}^{(1)})$ are not distinguishable, because both $LS_{MD}^{(1)}$ and $TS_{MD}^{(1)}$ are constructed in the same way with $T_A(-, \mathcal{L}_f^{MD}(LS))$ and thus their empirical distributions are the same. The graphs are consistent again with **[Hyp-1]**.

5 Discussions

5.1 Neuron Coverage

As explained in Section 3.1, the distortion relation ($-\preceq-$) between trained learning parameters is calculated in terms of observer functions. However, depending on the observer, the resultant distortion degree may be different. In an extreme case, a certain observer is not adequate to differentiate distortions. A question arises whether such distortion degrees are able to be measured directly. We will study *neuron coverage* [11] whether we can use it as such a measure.

A neuron is said to be *activated* if its output signal *out* is larger than a given threshold when a set of input signals in_j is presented; $out = \sigma(\sum w_j \times in_j)$. The weight values w_j s are constituents of trained W^* . *Activated Neurons* below refer

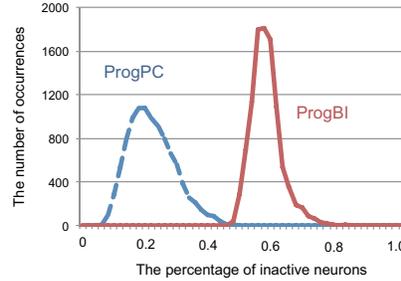


Fig. 5. Frequencies of Inactive Neurons

to a set of neurons that are activated when a vector data \mathbf{a} is input to a trained learning model as $y(W^*; \mathbf{a})$.

$$\text{Neuron coverage (NC)} = \frac{|\text{Activated Neurons}|}{|\text{Total Neurons}|}$$

In the above, $|X|$ denotes the size of a set X . Using this neuron coverage as a criterion is motivated by an empirical observation that different input-output pairs result in different degrees of neuron coverage [11].

Results of Experiment We focus on the neurons constituting the hidden layer in our NN learning model. As its activation function is *ReLU*, we choose 0 as the threshold. Figure 5 is a graph to show the numbers of input vectors leading to the chosen percentages of inactive neurons, $(1 - NC)$. These input vectors constitute the MNIST testing dataset TS of the size 10,000.

According to Figure 5, the graph for the case of ProgPC, the ratio of inactive neurons is almost 20%; namely, 80% of neurons in the hidden layer are activated to have effects on the classification results. However, the ProgBI graph shows that about 60% of them are inactive and do not contribute to the results. To put it differently, this difference in the ratios of inactive neurons implies that the trained learning model W_{BI}^* of ProgBI is distorted from W_{PC}^* of ProgPC, $W_{PC}^* \preceq W_{BI}^*$.

From the generation method of the distorted dataset, we have $LS \preceq LS_{PC}^{(1)}$ and $LS \preceq LS_{BI}^{(1)}$. $LS_{PC}^{(1)} \preceq LS_{BI}^{(1)}$ may also be satisfied, which is in accordance with the visual inspection of Figure 2. Furthermore, because of [Hyp-2] (Section 3.1), $W_{PC}^{(1)} \preceq W_{BI}^{(1)}$ is true. It is consistent with the situation shown in Figure 5 in that activated neurons in $W_{BI}^{(1)*}$ are fewer than those in $W_{PC}^{(1)*}$.

Figure 3 can be understood from a viewpoint of the neuron coverage. The empirical distribution of MNIST testing dataset TS is the same as that of MNIST training dataset LS . Because of the distortion relationships on training datasets, the distribution of TS is different from those of $LS_{MD}^{(1)}$ ($LS_{PC}^{(1)}$ or $LS_{BI}^{(1)}$). Moreover, Figure 3 shows that $|\text{Accuracy}(W_{PC}^{(1)*}, TS) - \text{Accuracy}(W_{PC}^{(0)*}, TS)|$ is

smaller than $|Accuracy(W_{PC}^{(1)*}, TS) - Accuracy(W_{PC}^{(0)*}, TS)|$. Therefore, we see that the relationship $LS_{PC}^{(1)} \preceq LS_{BI}^{(1)}$ is satisfied. Because of **[Hyp-2]**, it implies $W_{PC}^{(1)*} \preceq W_{BI}^{(1)*}$. Therefore, the difference seen in Figure 3 is consistent with the situation shown in Figure 5.

In summary, the neuron coverage would be a good candidate as the metrics to quantitatively define the distortion degrees of trained learning models. However, because this view is based on the MNIST dataset experiments only, further studies are desirable.

5.2 Test Input Generation

We will see how the dataset or data generation method in Section 3.2 is used in software testing. Because the program is categorized as untestable [13], Metamorphic Testing (MT) [1] is now a standard practice for testing of machine learning programs. We here indicate that generating an appropriate data is desirable to conduct effective testing. In the MT framework, given an initial test data x , a translation function T generates a new follow-up test data $T(x)$ automatically.

For testing machine learning software, either \mathcal{L}_f (whether a training program is faithful implementation of machine learning algorithms) [7][17] or \mathcal{I}_f (whether an inference program show acceptable prediction performance against incoming data), generating a variety of data to show *Dataset Diversity* [9] is a key issue. The function T_A introduced in Section 3.2 can be used to generate such a follow-up *dataset* used in the MT framework [10]. In particular, corner-case testing would be possible by carefully chosen such a group of biased datasets.

DeepTest [15] employs Data Augmentation methods [5] to generate test data. Zhou and Sun [19] adopts generating fuzz to satisfy application-specific properties. Both works are centered around generating test data for negative testing, but do not refer to statistical distribution of datasets.

DeepRoad [18] adopts an approach with Generative Adversarial Networks (GAN) [2] to synthesize various weather conditions as driving scenes. GAN is formulated as a two-player zero-sum game. Given a dataset whose empirical distribution is ρ^{DS} , its Nash equilibrium, solved with Mixed Integer Linear Programming (MILP), results in a DNN-based generative model to emit new data to satisfy the relation $\mathbf{x} \sim_{i.i.d.} \rho^{DS}$. Thus, such new data preserve characteristics of the original machine learning problem. Consequently, we regard the GAN-based approach as a method to enlarge coverage of test scenes within what is anticipated at the training time.

Machine Teaching [20] is an inverse problem of machine learning, and is a methodology to obtain a dataset to optimally derive a *given* trained learning model. The method is formalized as a two-level optimization problem, which is generally difficult to solve. We regard the machine teaching as a method to generate unanticipated dataset. Obtained datasets can be used in negative testing.

Our method uses an optimization problem with one objective function for generating datasets that are not far from what is anticipated, but probably are biased to build up the dataset diversity.

6 Concluding Remarks

We introduced a notion of distortion degrees which would manifest themselves as faults and failures in machine learning programs, and studied the characteristics in terms of neuron coverages. However, further study would be needed how we rigorously measure the distortion degrees, which will make it possible to *debug* programs with the measurement results. If the measurement is light-weight and can be conducted for in-operation machine learning systems, we will be able to diagnose systems at operation time.

Acknowledgment

The work is supported partially by JSPS KAKENHI Grant Number JP18H03224, and is partially based on results obtained from a project commissioned by the NEDO.

References

1. T.Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T.H. Tse, and Z.Q. Zhou : Metamorphic Testing: A Review of Challenges and Opportunities, *ACM Computing Surveys* 51(1), Article No.4, pp.1-27, 2018.
2. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio : Generative Adversarial Nets, In *Adv. NIPS 2014*, pp.2672-2680, 2014.
3. I. Goodfellow, Y. Bengio, and A. Courville : *Deep Learning*, The MIT Press 2016.
4. S. Haykin : *Neural Networks and Learning Machines (3ed.)*, Pearson India 2016.
5. A. Krizhevsky, I. Sutskever, and G. E. Hinton: Imagenet classification with deep convolutional neural networks. In *Adv. NIPS 2012*, pp. 1097-1105, 2012.
6. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner: Gradient-based learning applied to document recognition, In *Proceedings of the IEEE*, 86(11), pp.2278-2324, 1998.
7. S. Nakajima and H.N. Bui : Dataset Coverage for Testing Machine Learning Computer Programs, In *Proc. 23rd APSEC*, pp.297-304, 2016.
8. S. Nakajima : Quality Assurance of Machine Learning Software, In *Proc. IEEE GCCE 2018*, pp.601-604, 2018.
9. S. Nakajima : Dataset Diversity for Metamorphic Testing of Machine Learning Software, In *Post-Proc. 8th SOFL+MSVL*, pp.21-38, 2018.
10. S. Nakajima and T.Y. Chen: Generating Biased Dataset for Metamorphic Testing of Machine Learning Programs, In *Proc. IFIP-ICTSS 2019*, pp.56-64, 2019.
11. K. Pei, Y. Cao, J. Yang, and S. Jana : DeepXplore: Automated Whitebox Testing of Deep Learning Systems, In *Proc. 26th SOSP*, pp.1-18, 2017.
12. J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N.D. Lawrence (eds.) : *Dataset Shift in Machine Learning*, The MIT Press 2009.
13. S. Segura, D. Towey, Z.Q. Zhou and T.Y. Chen: Metamorphic Testing: Testing the Untestable, *IEEE Software* (in press).
14. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruma, D. Erhan, I. Goodfellow, and R. Fergus : Intriguing properties of neural networks, In *Proc. ICLR*, 2014.
15. Y. Tian, K. Pei, S. Jana, and B. Ray : DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars, In *Proc. 40th ICSE*, pp.303-314, 2018.

16. D. Warde-Farley and I. Goodfellow: Adversarial Perturbations of Deep Neural Networks, in *Perturbation, Optimization and Statistics*, The MIT Press 2016.
17. X. Xie, J.W.K. Ho, C. Murphy, G. Kaiser, B. Xu, and T.Y. Chen : Testing and Validating Machine Learning Classifiers by Metamorphic Testing, *J. Syst. Softw.*, 84(4), pp.544-558, 2011.
18. M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid: DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems, In *Proc. 33rd ASE*, pp.132-142, 2018.
19. Z.Q. Zhou and L. Sun: Metamorphic Testing of Driverless Cars, *Comm. ACM*, vol.62, no.3, pp.61-67, 2019.
20. X. Zhu : Machine Teaching: An Inverse Problem to Machine Learning and an Approach Toward Optimal Education, In *Proc. 29th AAAI*, pp.4083-4087, 2015.