Master's Thesis

Title

On an adversarial example attack by partially monitoring and tampering input features to machine learning models; its possibilities and countermeasures

Supervisor
Professor Masayuki Murata

Author
Yukiko Tsunoda

January 29th, 2021

Department of Information Networking
Graduate School of Information Science and Technology
Osaka University

Master's Thesis

On an adversarial example attack by partially monitoring and tampering input features to machine learning models; its possibilities and countermeasures

Yukiko Tsunoda

Abstract

Machine learning has become used in various areas such as anomaly detection on factories and farms. As machine learning becomes popular, machine learning models become the target of attacks. Adversarial examples are one of the most serious attacks for machine learning. In this attack, the attacker adds small perturbations to the input features of a machine learning model. By adding such small perturbations, the attacker makes the machine learning model make wrong decisions. Many researchers have demonstrated the possibility of adversarial examples. Though they discussed countermeasures, no fundamental solution has been found yet.

To generate adversarial examples, an attacker calculates the perturbation that causes the wrong decision of the target machine learning model. Most existing papers assume that attackers know the information of all input features and they calculate the perturbation based on the information. However, considering the case that a machine learning model is used to identify the current status from the information obtained from multiple sensors, attackers hack only a limited number of sensors that have vulnerabilities and monitor the features from the hacked sensors. If attackers can make the machine learning model make wrong decisions even in such cases, the risk of the adversarial examples is high.

In this thesis, we discuss the possibility of an adversarial example attack whereby an adversary can monitor and tamper a part of features. To demonstrate the possibility of the adversarial examples, we train a machine learning model to generate the adversarial examples. In this model, we use a part of features that can monitor by the attacker as the input. The output of this model is the features to cause the wrong decision of the target model. By using this model, we demonstrate that attackers can make machine learning models make wrong decisions even if they can monitor and tamper a part of features.

Moreover, we also propose the countermeasure against the adversarial examples by the attackers who can monitor and tamper a part of features. This method is based on that the attackers can monitor and tamper only a part of features. In this method, we first train an additional model called a *feature-removed model* that can make similar decisions without specified features. When the machine learning model makes a decision, we identify the important features for the decision and obtain the output of the model with feature loss by eliminating the important features. Then, we detect attacks by comparing the outputs of the original model and the model with feature loss. In addition, if the attack is detected, we suggest the outputs of the model with feature loss as the candidates of the correct outputs.

We demonstrate the effectiveness of our countermeasure against the attack. As a result, we confirm that data causing the attack can be identified with a detection rate of 88% and a false-positive rate of 1%. The original results can be suggested with 96% as well.

Keywords

Adversarial example
Countermeasure methods
Security
Partially monitoring
Label prediction

Contents

List of Figures

List of Tables

1 Introduction

Machine learning has become used in various areas such as anomaly detection on factories and farms. Machine learning-based applications such as Smart Healthcare [1], Smart Home [2], and Smart Transportation enrich our lives. As machine learning becomes popular, machine learning models become the target of attacks. Adversarial examples are one of the most serious attacks for machine learning. In this attack, the attacker adds small perturbations to the input features of a machine learning model. By adding such small perturbations, the attacker makes the machine learning model make wrong decisions.

Many researchers have demonstrated the possibility of the adversarial examples [3–8]. The methods to detect the adversarial examples have also been proposed [9–13]. However, it was demonstrated that the adversarial examples that cannot be detected by the above methods exist [14, 15]. That is, no fundamental solution has been found yet.

To generate adversarial examples, an attacker calculates the perturbation that causes the wrong decision of the target machine learning model. Most existing papers assume that attackers know the information of all input features and they calculate the perturbation based on the information. However, considering the case that a machine learning model is used to identify the current status from the information obtained from multiple sensors, attackers hack only a limited number of sensors that have vulnerabilities and monitor the features from the hacked sensors. If attackers can make the machine learning model make wrong decisions even in such cases, the risk of the adversarial examples is high.

In this thesis, we discuss the possibility of an adversarial example attack in the case that an attacker can monitor and tamper a part of features. To demonstrate the possibility of the adversarial examples, we train a machine learning model to generate the adversarial examples. In this model, we use a part of features that can monitor by the attacker as the input. The output of this model is the features to cause the wrong decision of the target model. By using this model, we demonstrate that attackers can make machine learning models make wrong decisions even if they can monitor and tamper a part of features.

Moreover, we also propose the countermeasure against the adversarial examples by the attackers who can monitor and tamper a part of features. This method is based on that the attackers can monitor and tamper only a part of features. In this method, we first train an additional model called a *feature-removed model* that can make similar decisions without specified features. When the machine learning model makes a decision, we identify the important features for the decision and obtain the output of the model with feature loss by eliminating the important features. Then, we detect attacks by comparing the outputs of the original model and the model with feature loss. In addition, if the attack is detected, we suggest the outputs of the model with feature loss as the candidates of the correct outputs.

2 Related Work

2.1 Adversarial Example

Szegedy et al. [3] showed the first method that makes neural network model misclassify labels by adding perturbation to the input features. In their approach, they added a small perturbation based on the gradient. As a result, they demonstrated that a small perturbation that cannot be recognized by a human can cause a wrong decision of a neural network.

Goodfellow et al. [4] proposed a method to generate adversarial examples called the fast gradient sign method (FGSM) method. In this method, the perturbation is calculated based on the sign of the gradient of the loss function. This method generates the perturbation that increases the loss of the target model.

Moosavi-Dezfooli et al. proposed another method to generate adversarial examples called DeepFool [5]. In this method, an adversarial example is generated by iteratively updating the perturbation based on the gradient of the loss function. Moosavi-Dezfooli et al. also demonstrated that a universal and very small perturbation vector that causes natural images to be misclassified with high probability exists.

Several papers demonstrated that adversarial examples can be generated from the physical world. Adversarial patch [6] is a method to generate patches that cause misrecognition of the machine learning model using the camera images. This method generates a patch that can be printed. By adding such a printed image, the machine learning model misclassifies the camera images.

Eykholt et al. [7] demonstrated the adversarial examples in the realistic scenario. They generated the images to cause the misclassification of the traffic sign and stuck the printed images to the traffic sign. They demonstrated that such attacks can cause misrecognition from a stop sign to a speed-limit sign. Sharif et al. [8] demonstrated a similar attack that causes misidentification of a person by wearing a specific accessory such as glasses.

Most researches on adversarial examples assume that the attackers know the information of all input features and they calculate the perturbation based on the information. However, considering the case that a machine learning model is used to identify the current status from the information obtained from multiple sensors, attackers hack only a limited number of sensors that have vulnerabilities and monitor the features from the hacked sensors. If attackers can make the machine learning model make wrong decisions even in such cases, the risk of the adversarial example is high. Therefore, in this thesis, we focus on the case that an adversarial example attack in the case that an attacker can monitor and tamper a part of features.

2.2 Adversarial Examples Detection

There are several methods to detect adversarial examples.

Metzen et al. proposed a method to detect adversarial examples [9]. They created a small neural network model to detect adversarial examples.

Safetynets is another method to detect adversarial examples [10]. This method trains SVM classifiers to detect attacks by using the activation patterns of the nodes in the target model.

Feinman et al. proposed a detection method based on that the uncertainty of adversarial example is higher than the data without attacks [11]. They made Bayesian neural networks to estimate the uncertainty of input data. Then based on the uncertainty, adversarial examples are detected.

Grosse et al. proposed a method to detect adversarial examples from the statistical difference between the data with and without perturbations [12]. Hendrycks et al. used the principal component analysis to detect adversarial examples. Meng et al. proposed a framework against adversarial examples [13]. In this framework, adversarial examples are detected based on the distribution of the training data. Then, if adversarial examples are detected, the framework tries to reconstruct the input so that the target classifier can make correct decisions.

However, Carlini and Wagner demonstrated that the adversarial examples that cannot be detected by the above methods exist [14, 15]. That is, no fundamental solution has been found yet.

In this thesis, we propose a countermeasure against adversarial examples, focusing on the case that the attackers can monitor and tamper a limited number of features. By focusing on the case, we propose a method to detect attacks by using features that cannot be tampered by the attackers.

3 Adversarial Examples by Monitoring and Tampering a Part of Features

In this section, we present a method to generate adversarial examples by monitoring and tampering a part of features. We first describe the problem setting and notations. Then, we explain how to generate adversarial examples.

3.1 Problem Setting

Target Model In this thesis, we focus on the classifier that outputs labels corresponding to input features. We assume the input features can be separated into multiple groups. Each group includes the features that can be monitored and tampered at the same time. For example, features from a sensor can be tampered by hacking the sensor. In this case the features from a sensor belong to the same group.

Ability of Attackers In this thesis, we assume that the attacker has enough information on the target model. That is, the attacker knows the information on the architecture and parameters of the target model. In addition, we assume that the attacker also knows enough training data that are used to train the target model. However, when generating the adversarial examples, the attackers can monitor and tamper only the features in one of the feature groups. That is, the attacker cannot obtain the features outside the group he/she can monitor. This assumption corresponds to the condition that the attacker can monitor and tamper the features only from the hacked sensors in the case that the target model uses the features from multiple sensors.

Goal of Attacks In this thesis, we focus on the target attacks. That is, the attacker tries to make the target model output the label he/she wants.

3.2 Notation

We denote $f()$ as the target machine learning model. $f(x)$ is the output of the model when the input features are x.

We denote X^{now} as the actual feature values when attacks. We introduce a vector M to represent the features that can be monitored and tampered by the attacker; the ith element of M is set to 1 if the ith feature can be monitored and tampered by the attacker, 0 otherwise. That is, the attacker can obtain $M^T X^{now}$.

We also introduce a function g that represents the function to generate the features generated by the attackers. That is, the attacker generates $g(M^T X^{now})$ and rewrite $M^T X^{now}$ to $g(M^T X^{now})$. As a result, the input of the target machine learning model becomes $(I - M)^T X^{now} + M^T g(M^T X^{now})$ where I is the vector whose all elements are 1.

3.3 Generation of Adversarial Examples

3.3.1 Training of Generative Model

In this thesis, the attacker generates the adversarial examples by using the function $g()$. The function $g()$ is a generative model trained by the information the attacker has. In this section, we explain how to train $g()$, when the attacker has the information on the target model $f()$ and the dataset to used to train $f()$ and M is fixed.

In this thesis, we construct $g()$ as a neural network. The size of the input features of $g()$ is the number of elements of M whose values are 1. The size of the output vector of $g()$ is also the same.

When training $g()$, we connect output of $g()$ to $f()$ as shown in Figure 1. Then we define the loss function based on the output of $f()$. We use the loss function defined as

$$L_g = L(f((1 - M)^T X^{now} + M^T g(M^T X^{now})), t'). \tag{1}$$

where t' is the label the attacker wants the target model to output, and $L(t, t')$ is the loss function defined between the output label t and the target label t'. We can use any loss functions for $L()$. In this thesis, we use the softmax cross-entropy defined as $L()$, because the softmax cross-entropy is widely used for training the neural networks for classification tasks.

The attacker trains $g()$ by using the training data he/she has. The training data includes all features including the features that cannot be monitored by the attacker when generating the attack. Thus, the attacker can calculate $f((1-M)^T X^{now} + M^T g(M^T X^{now}))$ for training data.

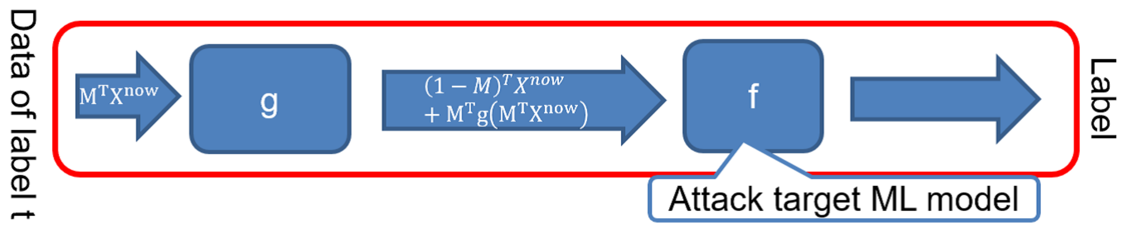

Figure 1: Attack generation image

We train $g()$ by using the training data and the loss function defined above. $g()$ can be trained by the same manner as the general training of neural networks. In this thesis, we use the stochastic gradient descent (SGD) method to train $g()$ so that the loss function is minimized.

3.3.2 Generation of Adversarial Examples

The attacker generates adversarial examples by using $g()$.

The attacker monitors the features that he/she can monitor. The monitored features can be represented as $M^T X^{now}$. Then, he/she obtain how to tamper the features that he/she can tamper by $g(M^T X^{now})$. Finally, he/she change the ith feature value among the features can be tampered to ith value of $g(M^T X^{now})$. As a result the input features of the target machine learning model $f()$ becomes $(1-M)^T X^{now} + M^T g(M^T X^{now})$.

In the above steps, we can obtain the output of $g()$ immediately. That is, the attackers can tamper the features in real-time.

4 Countermeasures

In this section, we propose a countermeasure against adversarial examples described in the previous section. We first describe the overview of the proposed countermeasure. Then, we explain each step of our countermeasure.

4.1 Overview of Countermeasure

In this thesis, we propose a countermeasure, focusing on the features that cannot be changed by the attacker. In the attack discussed in this thesis, an attacker can monitor and tamper only features in a group. Even if the attacker changes all the features in the group, we have the remaining features outside the group that cannot be changed by the attacker. The remaining features may include the information on the actual label. That is, it is possible to obtain the true label by performing the classification without using the features that may be tampered by the attacker.

Based on the above idea, we propose a countermeasure against adversarial examples. In our countermeasure, When the output of the target machine learning model is obtained, we first identify the candidate of feature groups that may be tampered. Then, we obtain the classification results without using features in each of the identified feature groups. Finally, we detect adversarial examples by comparing the results from the target model and the results from the classification without using the identified features. In addition, if the adversarial examples are detected, we can present the possible true labels by using the results of the classification without using the identified features.

The rest of this section explains the steps in our countermeasure.

4.2 Identification of Tampered Features

In the adversarial examples, the tampered features have a significant impact on the classification results; if the features have not tampered, the classification results must be the true label and different from the results of the classification with tampered features. Therefore, the tampered features can be identified by identifying the features with a large impact on the classification results.

There are several methods to identify the features with a large impact on the results.

We can apply any methods. In this thesis, we use SmoothGrad [16]. SmoothGrad is one of the methods to clarify the reason for classification in deep learning.

The SmoothGrad focuses on the impact of the change of the feature values on the loss function of the target model. If the value of the loss function changes significantly by changing a value of a feature, the feature has a large impact on the results. Such impacts can be evaluated by partial derivative; the impact of the feature can be evaluated by the partial derivative of the loss function with respect to a feature. But just by using the partial derivative, we cannot consider the correlation with other features. Therefore, the SmoothGrad calculates the average of values of the partial derivative after adding the noises to features. That is, the impact of the ith feature on the results $\hat{M}_c(x, i)$ is obtained by

$$\hat{M}_c(X, i) \quad = \quad \frac{1}{n}\sum_{1}^{n} M_c(X + N(0, \sigma), i), \tag{2}$$

where n means the number of samples, X means the input features, $N(0, \sigma)$ means the noise that follows the Gaussian distribution whose mean and variance are 0 and σ. $M_c(X, i)$ is defined by

$$M_c(X, i) = \frac{\partial f(X)}{\partial x_i} \tag{3}$$

where $f()$ is the loss function and x_i is the ith feature.

In our countermeasure, we calculate $\hat{M}_c(X, i)$ for each features. The features whose absolute values of $\hat{M}_c(X, i)$ are large have a large impact on the classification result. Therefore, we identify the feature group which includes the features with large $\hat{M}_c(X, i)$ as the feature group that may be tampered by the attacker. In this thesis, we calculate $\hat{M}_c^g(X)$ defined by

$$\hat{M}_c^g(X) = \sum_{i \in I_g} \frac{|\hat{M}_c(X, i)|}{\max_i \hat{M}_c(x, i)} \tag{4}$$

where I_g is the set of the IDs of the features in the group g. Then we identified the feature group g if $\hat{M}_c^g(X)$ exceeds a threshold.

4.3 Classification by Excluding Identified Features

In our countermeasure, we obtain the classification results without using the features in each of the identified groups. To obtain such results, we introduce an additional model called *feature-removed model.*

We construct the feature-removed model so that we can set the features to be excluded and obtain the classification results without using the set features. If some important features are excluded, the feature-removed model cannot identify the label. Considering this, the feature-removed model outputs multiple possible labels instead of identifying a single label. By using the feature-removed model, we can obtain the possible labels, considering the possibility of the adversarial examples by setting the identified features as the features to be excluded.

The rest of the subsection explains the structure of the feature-removed model and how to train the feature removed model.

4.3.1 Structure of the Feature-removed Model

In this thesis, we use a neural network to construct the feature-removed model. We can use any neural network structures such as fully-connected neural networks and convolutional neural networks. The difference from the existing neural networks is only a mechanism to handle excluded features.

The feature-removed model accepts the same input as the target model but we can set the features to be excluded. In the feature-removed model, the excluded features are set to 0.

In the middle layer of the feature-removed model, the output of the ith node of jth layer $o_{i,j}$ is calculated by

$$o_{i,j} = a\left(\sum_{k\in C_{i,j}} \frac{1}{1 - N^{\text{excluded}}_{C_{i,j}}} w_{j-1,k,i} o_{k,j-1} + b_{i,j}\right) \tag{5}$$

where $a()$ is the activation function, $C_{i,j}$ is the set of nodes in the $j-1$th layer connected to the ith node of jth layer, $w_{j-1,k,i}$ and $b_{i,j}$ are the weight and bias trained in the training phase. $N^{\text{excluded}}_{C_{i,j}}$ is the number of excluded features. When calculating $N^{\text{excluded}}_{C_{i,j}}$, we also regard the features in the middle layer that are calculated only by the excluded features as the excluded features. By using $N^{\text{excluded}}_{C_{i,j}}$, we can make the output of the middle layers without excluded features a similar scale to the output calculated by all features.

In the output layer of the feature-removed model, we use the Sigmoid function as an activation function instead of the softmax function. By using the Sigmoid function, we allow outputs for multiple labels to become 1.

4.3.2 Training of the Feature-removed Model

We train the feature-removed model so that the model can output possible labels without specified features. To train this model, we use the same dataset used to train the target model.

We train the model by randomly selecting the feature group to be excluded. By doing so, we construct the model that can work excluding the specified features. The output of the model should include the training labels. But we can allow that the output includes the different labels. Considering this, we use the following loss function to train the feature-removed model.

$$L^{feature-removed}(Y, T) = -\sum_i \left(w(t_i)(t_i \log y_i + (1 - t_i) \log(1 - y_i))\right) \tag{6}$$

where Y is the output of the model, T is the training label, y_i is the ith element of Y and t_i is the $i$the element of T. t_i is set to 1 if the training label is i, otherwise 0. $w(t)$ is the weight defined for t. We set $w(0) << w(1)$ so as to include the training label in the output.

4.4 Decision

After obtaining the outputs of the feature-removed model, we compare the outputs of the feature-removed model and the target model to detect adversarial examples. If multiple feature groups are identified as the feature groups that may be tampered, we compare the outputs of the feature-removed model by excluding each of the groups.

If there exits a feature group whose corresponding outputs include a low probability for the output label of the original model, the output label is wrong considering the features excluding the feature group. In this case, we detect the attack. In addition, the true label is included in the outputs of the feature-removed model excluding the feature group. Thus, we present the outputs as the candidates of the true label.

If none of the feature groups have the outputs including a low probability for the output label of the original model but all feature groups include the other labels with high probability, it is possible that the true label is different from the original model. We label this case as "Suspicious". Then, we present the outputs of the feature-removed model as the candidate of the true label.

Otherwise, the output label of the original model matches the output of the feature-removed model. Thus, we label this case as "Normal".

Figure 2: MNIST dataset

5 Evaluation of Adversarial Examples and Their Countermeasures

5.1 Experimental Environment

5.1.1 Dataset

There are no open data from multiple sensors that can be used for evaluation in this thesis. Instead of the data from multiple sensors, we use the MNIST dataset, which is one of the most famous datasets in the field of machine learning and has been used for the demonstration of the adversarial examples [13]. The MNIST is an image dataset of handwritten numbers. The MNIST dataset consists of numeric images from "0" to "9" as shown in Figure 2. There are 60,000 training data and 10,000 test data, where each image is a 28×28 gray image and each pixel has a value between 0 and 255.

In this thesis, we set the feature groups by dividing each image into nine blocks as shown in Figure 3.

5.1.2 Target Model

In this evaluation, we construct a convolutional neural network to classify the MNIST dataset. The hyperparameters of the model are set according to the tutorial of TensorFlow.

5.2 Demonstration of Adversarial Examples

First, we demonstrate the adversarial examples by monitoring and tampering a part of features. In this demonstration, the attacker can monitor and tamper the features in one of the feature groups shown in Figure 3.

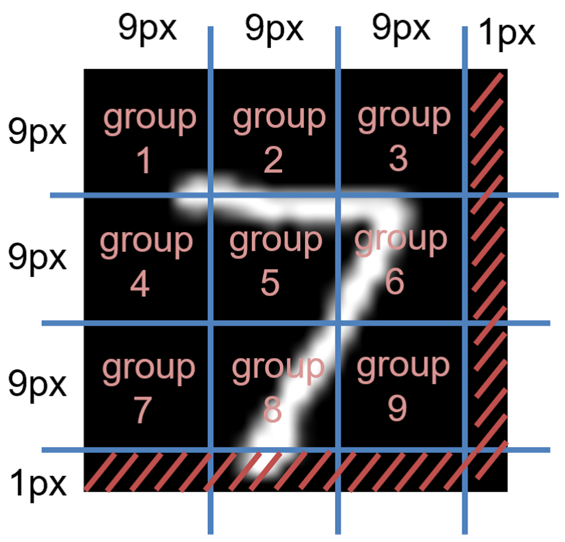

Figure 3: Definition of feature groups in our evaluation

Table 1: Attack success ratio of attacks to make the model misclassify 1 as 7

0.007	0.974	0.000
0.006	0.989	0.199
0.004	0.208	0.008

In our evaluation, there are 10 kinds of pre-attack labels, 9 kind of target labels of attacks and 9 attack target group. That is, $10 \times 9 \times 9 = 810$ patterns of attacks. We generate attacks for all patterns and calculate the attack success ratio defined by

$$\epsilon = \frac{Y}{X}. \tag{7}$$

where X is the number of generated images and Y is the number of images that are classified into the target label.

Table 1 shows the attack success ratio of attacks to make the model misclassify 1 as 7. The ratio shown in the table indicates the attack success ratio for the case that each feature group is monitored and tampered. This table shows that the attack success ratio depends on the feature groups. But by monitoring and tampering some feature groups, the attack succeeds with high probability.

Table 2 shows the attacks with a high attack success ratio. The table shows that many attacks succeed with a success ratio higher than 0.9. That is, the attacker that can monitor and tamper only a limited number of features can generate attacks that cause misclassification.

Table 2 shows that the success ratio becomes high when the 5th group can be monitored and tampered. The 5th group is in the center of the image. Thus, the attacker can obtain enough information to generate adversarial examples from the features in the 5th group. In addition, the features of the 5th group have also a large impact on the target model. As a result, the attacker that monitors and tamper features in the 5th group achieves a high attack success ratio.

The environment settings for experimental evaluation, i.e., structure hyperparameters of a machine learning model, are described below.

Table 2: Attacks with a high success ratio

attack target group	pre-attack label	target label	success ratio
2	1	7	0.974
2	9	4	0.733
5	0	8	0.858
5	1	3	0.977
5	1	4	0.985
5	1	5	0.948
5	1	6	0.964
5	1	7	0.989
5	1	8	0.997
5	1	9	0.957
5	2	8	0.727
5	3	5	0.708
5	3	8	0.705
5	4	7	0.925
5	6	5	0.784
5	7	3	0.741
5	7	8	0.764
5	7	9	0.904
5	9	7	0.728
5	9	8	0.765
8	1	2	0.978
8	1	8	0.989
8	7	2	0.829
8	9	8	0.918

	Attack	Suspicious	Normal	Total
group2	13	7	0	20
group5	166	14	0	180
group8	33	5	2	40
total	212	26	2	240
original data	1	40	59	100

Table 3: Accuracy of attack detection

5.3 Evaluation of Countermeasure

We evaluate our countermeasure. In this evaluation, we generate attacks whose success ratio is high by setting the targets as shown in Table 2. For each setting, we generate 10 attacks and evaluate the accuracy of the detection. In addition, we evaluate our method by using the 100 data without attacks. In this evaluation, we set the threshold to identify the feature groups to 12.

Table 3 shows the results grouped by the feature group that is monitored and tampered by the attacker. Our method outputs three kinds of attack levels; attack, suspicious and normal. Table 3 shows that more than 88 % of attacks are detected as attacks. Only less than 1 % of attacks are not detected. That is, our method detects attacks accurately. On the other hand, the ratio of the original data that are mistakenly detected is 1 %. However, 40 % of the original data are labeled as suspicious. This is because some feature groups are necessary to identify the specific label. As a result, the feature-removed model outputs multiple candidates, and we cannot identify whether the data is an attack or normal. To avoid this problem, we need a more sophisticated method, which is one of our future work.

We also evaluate the output labels of our method. Table 4 shows the result. Our method may output multiple labels. Thus, we divided their cases. The first case is the case that the number of the output labels is 1 and the label matches the correct label. In Table 4, we call this case *fully success*. The second case is the case that our method outputs multiple labels that include the correct label. In Table 4, we call this case *partially success*. The last case is the case that no output labels match the correct label. Table 4, we call this case *wrong*.

Table 4 shows that our method output correct labels with high accuracy. In 230 out of 240 cases, our method outputs correct labels. Moreover, in 132 cases, our method can identify the single correct label. That is, by using our method, we can obtain the correct output even when the attackers try to generate adversarial examples.

Table 4: Number of cases that our method successfully outputs the correct label

attack target group	pre-attack label	post-attack label	Fully success	Partially success	Wrong
2	1	7	7	3	0
2	9	4	1	3	6
5	0	8	8	2	0
5	1	3	0	10	0
5	1	4	8	2	0
5	1	5	6	4	0
5	1	6	4	6	0
5	1	7	8	2	0
5	1	8	8	2	0
5	1	9	6	4	0
5	2	8	8	2	0
5	3	5	6	4	0
5	3	8	8	2	0
5	4	7	7	3	0
5	6	5	7	3	0
5	7	3	6	4	0
5	7	8	6	4	0
5	7	9	4	6	0
5	9	7	1	9	0
5	9	8	1	9	0
8	1	2	6	4	0
8	1	8	5	4	1
8	7	2	6	4	0
8	9	8	5	2	3
total			132	98	10

6 Conclusion and Future Work

In this thesis, we discussed the possibility of an adversarial example attack whereby an adversary can monitor and tamper a part of features. To demonstrate the possibility of the adversarial examples, we trained a machine learning model to generate the adversarial examples. In this model, we use a part of features that can monitor by the attacker as the input. The output of this model is the features to cause the wrong decision of the target model. By using this model, we demonstrated that attackers can make machine learning models make wrong decisions even if they can monitor and tamper a part of features.

Moreover, we also proposed the countermeasure against the adversarial examples by the attackers who can monitor and tamper a part of features. This method is based on that the attackers can monitor and tamper only a part of features. In this method, we first train an additional model called a *feature-removed model* that can make similar decisions without specified features. When the machine learning model makes a decision, we identify the important features for the decision and obtain the output of the model with feature loss by eliminating the important features. Then, we detect attacks by comparing the outputs of the original model and the model with feature loss. In addition, if the attack is detected, we suggest the outputs of the model with feature loss as the candidates of the correct outputs.

We demonstrated the effectiveness of our countermeasure against the attack. As a result, we confirm that data causing the attack can be identified with a detection rate of 88% and a false-positive rate of 1%.

We demonstrated attacks and countermeasures using MNIST in this thesis. However, the discussion in this thesis is applicable to the other cases. Especially, the attack scenario discussed in this paper is important of the machine learning model using multiple sensors. Therefore, we need to demonstrate the attacks and evaluate our countermeasure in the case of the sensor-based machine learning model, which is one of our future work.

Acknowledgments

This thesis would not accomplish without a lot of great supports from several people. First, I would like to express my deepest gratitude to Professor Masayuki Murata of Osaka University, for providing me with his enormous support and insightful comments that were valuable during my study. His continued support, including insightful advice and meticulous comments, was essential to this research. Also, I'd like to express my deepest thanks to Associate Professor Yuichi Ohsita of Osaka University for his valuable advice, technical support, and constructive discussions that contributed to the progress of my research. He devotes a lot of time on gave me much advice on my research. Without his support, my work would not have been accomplished. I am grateful to Associate Professor Shin'ichi Arakawa and Assistant Professor Daichi Kominami of Osaka University for providing me many valuable comments. Moreover, I also would like to thank Assistant Professor Naoto Yanai of Osaka University. He gives insightful comments and suggestions. Their advice was always informative and helpful. In addition, I would like to thank all the members of the laboratory for support and meaningful discussion about my research and hearty encouragement. And last but not the least, I would like to express my deepest gratitude to my family for their wonderful support.

References

[1] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings Bioinform.*, vol. 19, no. 6, pp. 1236–1246, 2018. [Online]. Available: https://doi.org/10.1093/bib/bbx044

[2] H. Jiang, C. Cai, X. Ma, Y. Yang, and J. Liu, "Smart home based on wifi sensing: A survey," *IEEE Access*, vol. 6, pp. 13 317–13 325, 2018. [Online]. Available: https://doi.org/10.1109/ACCESS.2018.2812887

[3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.6199

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[5] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016.* IEEE Computer Society, 2016, pp. 2574–2582. [Online]. Available: https://doi.org/10.1109/CVPR.2016.282

[6] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *CoRR*, vol. abs/1712.09665, 2017. [Online]. Available: http://arxiv.org/abs/1712.09665

[7] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June*

18-22, 2018. IEEE Computer Society, 2018, pp. 1625–1634. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Eykholt_Robust_Physical-World_Attacks_CVPR_2018_paper.html

[8] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1528–1540. [Online]. Available: https://doi.org/10.1145/2976749.2978392

[9] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," *CoRR*, vol. abs/1702.04267, 2017. [Online]. Available: http://arxiv.org/abs/1702.04267

[10] J. Lu, T. Issaranon, and D. A. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 446–454. [Online]. Available: https://doi.org/10.1109/ICCV.2017.56

[11] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," *CoRR*, vol. abs/1703.00410, 2017. [Online]. Available: http://arxiv.org/abs/1703.00410

[12] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. D. McDaniel, "On the (statistical) detection of adversarial examples," *CoRR*, vol. abs/1702.06280, 2017. [Online]. Available: http://arxiv.org/abs/1702.06280

[13] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 135–147. [Online]. Available: https://doi.org/10.1145/3133956.3134057

[14] N. Carlini and D. A. Wagner, “Adversarial examples are not easily detected: Bypassing ten detection methods,” in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*, B. M. Thuraisingham, B. Biggio, D. M. Freeman, B. Miller, and A. Sinha, Eds. ACM, 2017, pp. 3–14. [Online]. Available: https://doi.org/10.1145/3128572.3140444

[15] ——, “Magnet and” efficient defenses against adversarial attacks” are not robust to adversarial examples,” *CoRR*, vol. abs/1711.08478, 2017. [Online]. Available: http://arxiv.org/abs/1711.08478

[16] D. Smilkov, N. Thorat, B. Kim, F. B. Viégas, and M. Wattenberg, “Smoothgrad: removing noise by adding noise,” *CoRR*, vol. abs/1706.03825, 2017. [Online]. Available: http://arxiv.org/abs/1706.03825