Unsupposable Test-data Generation for Machine-learned Software

Naoto Sato[1,a)] Hironobu Kuruma[1] Hideto Ogawa[1]

Abstract: As for software development by machine learning, a trained model is evaluated by using part of an existing dataset as test data. However, if data with characteristics that differ from the existing data is input, the model does not always behave as expected. Accordingly, to confirm the behavior of the model more strictly, it is necessary to create data that differs from the existing data and test the model with that different data. The data to be tested includes not only data that developers can suppose (*supposable* data) but also data they cannot suppose (*unsupposable* data). To confirm the behavior of the model strictly, it is important to create as much unsupposable data as possible. In this study, therefore, a method called "unsupposable test-data generation" (UTG)—for giving suggestions for unsupposable data to model developers and testers—is proposed. UTG uses a variational autoencoder (VAE) to generate unsupposable data. The unsupposable data is generated by acquiring latent values with low occurrence probability in the prior distribution of the VAE and inputting the acquired latent values into the decoder. If unsupposable data is included in the data generated by the decoder, the developer can recognize new unsupposable features by referring to the data. On the basis of those unsupposable features, the developer will be able to create other unsupposable data with the same features. The proposed UTG was applied to the MNIST dataset and the House Sales Price dataset. The results demonstrate the feasibility of UTG.

1. Introduction

In recent years, software developed by machine learning has been introduced in various fields of industry. In conjunction with this trend, techniques for checking the behavior of such software developed by machine learning have been developed. Some examples of proposed techniques verify deep neural networks (DNNs) and ensemble models using decision trees to determine whether their input data and output data satisfy certain properties [1] [2] [4] [5]. Hereafter, software developed by machine learning (that is, trained software) is referred to as a *model* in this paper. By using those verification techniques, it is possible to verify exhaustively whether a model satisfies requirements regarding safety, for example. However, it is known that exhaustive verification cannot be completed within a practical time when the scale of the model is largeMoreover, even if the correctness of the model cannot be defined as a property, these exhaustive verification techniques cannot be used. For example, in the case of an image-recognition problem, the requirement that "A given image should be correctly identified as a person." cannot be defined as a property. In such cases, verifying the model by testing is effective.

As for development of a model, a certain part of an existing dataset is used as training data, and the rest is used as test data for evaluating the trained model. The generalizability of the model can be confirmed by using different data from the training data as the test data. With this method, it is possible to check that the developed model behaves as expected with existing data; however, if data with different characteristics from the existing data is input, the behavior does not always turn out as expected. Therefore, to confirm the behavior of the model more strictly, it is necessary to create input data differing from the existing data and test the model with that different data. However, even in the case of data differing from existing data, data that does not need to be supposed is excluded from the test data. This supposition-unrequired data is not always outside the domain of the input data. For example, in the case of a model that performs image recognition, the model accepts all images of a certain size as input data; however, the data that actually needs to be supposed is only a part of those images, that is, images of things that exist in the real world. Hereafter, the data that does not need to be supposed is called *unreal* data. Data that is not unreal data is referred to as *real data*. The boundary between unreal data and real data is not always clear. For example, in the case of an image-recognition model, the data that needs to be supposed comprises images of things that exist in the real world; however, it is difficult to define exactly what kind of images they are.

Furthermore, real data (that needs be supposed) can be classified as data that model developers and testers can easily suppose and data that is difficult to suppose. Hereafter, the former is called *supposable* data and the latter is called *unsupposable* data. Existing data is one part of supposable data. Whether certain data is supposable or unsupposable depends on the person supposing, so the boundaries between them are ambiguous. In the following, the person making suppositions about input data is referred to as

[1] Research & Development Group, Hitachi, Ltd., 292 Yoshida-cho, Totsuka-ku, Yokohama-shi, Kanagawa 244-0817, Japan
[a)] naoto.sato.je@hitachi.com

the *developer*.

From the above discussion, it can be said that although the boundaries between them are ambiguous, data can be classified as three types: supposable data, unsupposable data, and unreal data (Fig. 1). The data to be tested are supposable data and unsupposable data, of which the supposable data can be easily created. Therefore, to confirm the behavior of a model strictly, it is preferable to create as much unsupposable data as possible. Therefore, we propose "unsupposable test-data generation" (UTG) as a technique aiming to give suggestions for unsupposable data to developers. UTG uses a variational autoencoder (VAE) to generate data. As for a normal VAE, a latent value is sampled according to the prior distribution *P*, and data is generated by decoding the sampled latent value. Since the data generated in this way has similar characteristics to the existing data used for training the VAE, it is likely to be supposable data. On the other hand, with UTG, a latent value with a low probability of occurrence in *P* is acquired and decoded. Consequently, data with different characteristics from the existing data is generated. As described above, however, the boundaries between unreal data, supposable data, and unsupposable data are ambiguous, so the data generated by UTG is not always unsupposable data. Accordingly, UTG has parameters which are used to change the rarity of the acquired latent value. The developer changes the values of the parameters while referring to the generated data and exploratively determines the values of the parameters so that as much unsupposable data as possible is included in the data generated by the decoder. If unsupposable data is included in the data generated by the decoder, the developer can recognize unsupposable features by referring to that data. On the basis of those features, the developer can create other unsupposable data with those unsupposable features. In this study, by applying UTG to the MNIST dataset [24] and the House Sales Price dataset, it was confirmed that it is possible to generate unsupposable data from data generated by UTG. The first contribution of this paper is to propose UTG and show its implementation. The second contribution is to demonstrate the feasibility of UTG through case studies.

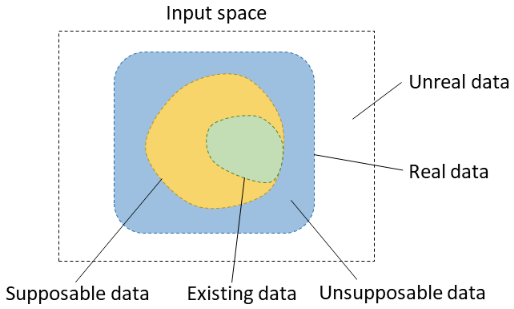


Fig. 1 Relationship between existing data, unreal data, supposable data, and unsupposable data

The rest of this paper is organized as follows. In Section 2, related research is described, and the position and aim of UTG are clarified. In Section 3, the variational auto-encoder (VAE) and the VQ (vector quantized)-VAE with PixelCNN, which are the basis for implementing UTG, are described. In Section 4, the UTG concept is explained, and the means of implementing UTG using VAE and VQ-VAE with PixelCNN is described. In Section 5, the results of the case study are presented, and in Section 6, those results are evaluated and discussed. In Section 7, the conclusions of this study are given, and future work is discussed.

2. Related Work

As approaches to create new data for testing a model, several methods that focus on the activation status of the neurons that make up a DNN are known [7] [6] [8]. As for these methods, new data is created by processing existing data so as to activate neurons that were not activated when the existing data was input. For example, in the case of image data, processing such as increasing the brightness or adding white noise is performed. Then, both the processed data and the unprocessed data are input to the model, and matching of the output values is confirmed. It has been shown experimentally that this method can demonstrate incorrect behavior of the model. A test method that inputs the data before and after processing into the model and compares and evaluates the output values in this way is called metamorphic testing [20] [21]. In an example in which metamorphic testing is applied to an objective-perception model of self-driving cars, small particles in the air and noise from sensors are given as effects [19]. Moreover, adding perturbation—instead of the semantic effects described above—to the extent that the original image data does not change significantly is also a way of processing [22].

The means of processing of the input data, such as increasing brightness, adding small particles, or adding perturbation, is defined according to the problem that the model solves. It is necessary to select how to process existing data so that the processed data becomes the data that should be supposed as the input data. For example, in the case of a model used in an environment in which brightness is strictly controlled, it is not necessary to suppose a change in brightness. On the other hand, in the case of a model that is used outdoors day and night, it is necessary to assume a change in brightness. The data that can be created by these methods is supposable data because it is created by processing existing data in a way that can be supposed by the developer

Moreover, using GAN [12] [13] or VAE [14] [15] [16], a natural image can be generated by changing some of the attributes from an existing image [9] [10] [11]. As for these methods, a neural network is trained by using a set of images having one attribute (A) and another set of images having another attribute (B). Then, when an arbitrary image with attribute A is input to the trained neural network, attribute A is deleted from that image, and an image with added attribute B is output instead. To generate data using these methods, the developer needs to define the attributes of the data to be generated. That is, the data that can be generated by these methods is taken as supposable data because it has the attributes that are expected by the developer. On the other hand, as for UTG, unsupposable data (which is difficult for the developer to expect) is generated without inputting a supposition by the developer such as the data-processing way, attributes, and so on.

3. Background

3.1 Variational Auto-encoder

A variational autoencoder (VAE) is composed of an encoder and a decoder. From z, which represents an unobservable feature of input data x, the decoder creates input data x' that may correspond to z. Here, z is called a latent variable and holds a K-dimensional vector value. Hereafter, the prior distribution of z is represented by P, which has a mean of zero, and the variance-covariance matrix is assumed to be a Gaussian distribution with identity matrix I. From input data x, the encoder creates a distribution of values of z from which x could have been generated. The distribution of z created by the encoder is assumed to be Gaussian. Here, the case in which $x0$ (corresponding to a certain $z0$) is given to the encoder is considered. $z1$ obtained by sampling the Gaussian distribution output by the encoder is expected to be a value close to $z0$. The value obtained by giving $z1$ to the decoder is taken as $x1$, which is expected to be similar to $x0$. To meet these expectations, the encoder and decoder are trained with the structure shown in Fig. 2. Hereafter, T represents the existing dataset that is used for training. The encoder accepts input data x and outputs mean μ and variance-covariance matrix σ, which are parameters of the Gaussian distribution. And z is obtained by sampling from the distribution determined by μ and σ. (Actually, instead of sampling, a method called reparameterization trick is used to calculate z from μ and σ by using random noise [14].) The decoder accepts z and outputs reconstructed data x'. The trained decoder is used for generating data (Fig. 3). The value of z is obtained by sampling the prior distribution P of z, and data g is generated by giving that value to the trained decoder. The distribution of g is similar to the distribution of T, so the decoder is used to generate data similar to T.

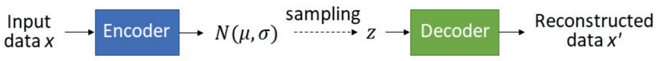


Fig. 2 Structure of VAE

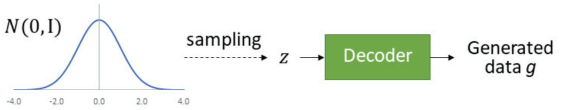


Fig. 3 Generating data by VAE

3.2 VQ-VAE with PixelCNN

When the target is image data, a VQ (vector quantized)-VAE [18] can be used to generate data with better image quality than that possible with a VAE. The structure of VQ-VAE is shown in Fig. 4. The VQ-VAE encoder accepts input data x and outputs z, which represents a $i \times j$ matrix whose elements are latent vector $z_{i,j}(1 \leq i \leq I, 1 \leq j \leq J)$. Latent vector $z_{i,j}$ forming z is a K-dimensional vector. $z_{i,j}$ that composes z is replaced with a vector of fixed values included in a list called a codebook. The latent vector obtained after replacement is called a discrete latent vector, which is represented as $\hat{z}_{i,j}$. The number of fixed value vectors included in the codebook—denoted by V—is finite. In this paper, the fixed-value vectors in the codebook are called *code vector* which are represented as $c[0], ..., c[v], ..., c[V-1]$. Among the code vectors, the one having the closest Euclidean distance to $z_{i,j}$ is selected as $\hat{z}_{i,j}$. As a result, latent vector $z_{i,j}$ is "quantized" to a discrete latent vector given as $\hat{z}_{i,j}$.

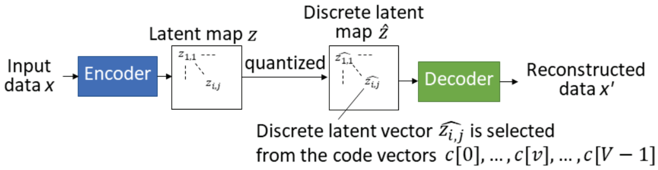


Fig. 4 Structure of VQ-VAE

As for VQ-VAE, an autoregressive model is used to estimate the prior distribution P and generate data. PixcelCNN is used by Oord, et al. [18] as an example of an autoregressive model for images, and they cite WaveNet as an example of an autoregressive model for audio. In this paper, we describe a method for generating image data using PixelCNN in Fig. 5.

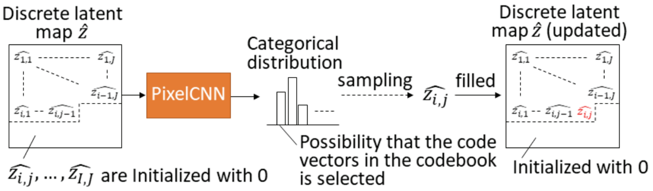


Fig. 5 PixelCNN over latents

PixelCNN accepts discrete latent map $\hat{z}$ in which values from $\hat{z}_{1,1}$ to $\hat{z}_{i,j-1}$ were filled as input. The categorical distribution for determining the value of $\hat{z}_{i,j}$ is then output. As described above, $\hat{z}_{i,j}$ is one of the code vectors included in the codebook. The categorical distribution output by PixelCNN represents the probability that each code vector included in the codebook will be selected as $\hat{z}_{i,j}$, the value of which is determined by sampling from the categorical distribution. When existing dataset T is input into the trained VQ-VAE encoder, discrete latent maps corresponding to T are obtained. These discrete latent maps ate then used to train PixelCNN.

As shown in Fig. 6, a discrete latent map can be generated by using the trained PixelCNN recursively. Then, the generated discrete latent map is input into the VQ-VAE decoder to generate data g. It is known that using VQ-VAE and PixelCNN makes it possible to generate image data with better image quality than that possible with VAE [18]. In particular, it is expected to prevent blurring of contours.

4. Unsupposable Test-data Generation(UTG)

4.1 Concept

As for UTG, unsupposable data is generated by acquiring a value with a low occurrence probability in prior distribution P of latent variable z. As described in Section 3.1, P is given as $N(0, I)$. As for usual data generation using a decoder, generated data g is obtained by sampling the value of z according to distribution P and giving it to the trained decoder. The dataset generated by the decoder is denoted by G ($g \in G$). The distribution of G is similar to that of existing dataset T used for training because the

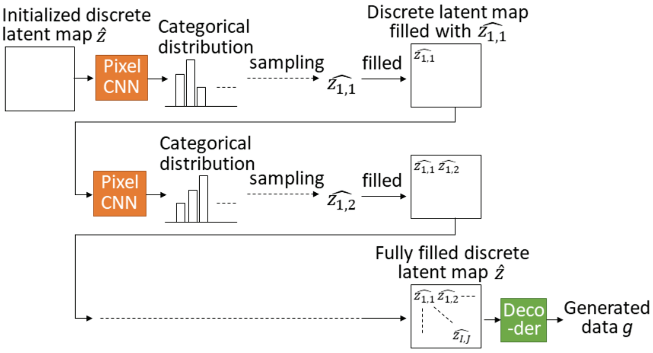

Initialized discrete latent map $\hat{z}$
Pixel CNN
Categorical distribution
sampling
$\widehat{z_{1,1}}$
filled
Discrete latent map filled with $\widehat{z_{1,1}}$
$\widehat{z_{1,1}}$
Pixel CNN
Categorical distribution
sampling
$\widehat{z_{1,2}}$
filled
$\widehat{z_{1,1}}$ $\widehat{z_{1,2}}$
Fully filled discrete latent map $\hat{z}$
$\widehat{z_{1,1}}$ $\widehat{z_{1,2}}$
$\widehat{z_{I,J}}$
Deco -der
Generated data g

Fig. 6 Generating data by VQ-VAE with PixelCNN

encoder and decoder are optimized for T. Here, it is assumed that the value of z obtained from P has a high occurrence probability, that is, its absolute deviation in P is small. Since z represents an unobservable feature of x, it can be said that g generated from z is likely to have features with high occurrence probability in T. On the contrary, when z obtained from P has a low occurrence probability, it can be said that the generated g is likely to have features which are not commonly observed in T. In other words, g might contain unsupposable features.

Utilizing this fact, UTG intentionally acquires a latent value with a low occurrence probability in P, which is denoted z^u. The acquired latent value z^u is input into the decoder to generate data g^u. It is considered that the dataset generated by this method is more likely to contain unsupposable data than one generated from z obtained according to P. The dataset generated by this method is hereafter called a *likely-unsupposable* dataset LU. By referring to LU, a developer can get suggestions for unsupposable data. However, as described in Section 1, the boundaries between supposable data, unsupposable data, and unreal data are ambiguous and depend on the developer, so this method cannot always generate unsupposable data. Even if the boundaries between supposable data, unsupposable data, and unreal data are known, the lowest probability of occurrence in P that can be adopted as z^u to generate unsupposable data also depends on the results of training the VAE. Therefore, as for UTG, the method of obtaining z^u from P is parameterized, and the rarity of z^u to be obtained can be changed by changing the values of the parameters. The developer changes the values of the parameters while referring to the generated dataset LU and adjusts them so that LU contains a lot more unsupposable data. By adjusting the occurrence probability of the acquired z^u in P in this manner, the possibility of generating unsupposable data is improved exploratively.

4.2 Implementation on VAE

The concept of UTG described in Section 4.1 is to (i) "acquire z^u with low occurrence probability in priority distribution P and generate g^u" and (ii) "to exploratively adjust the occurrence probability in P of z^u by changing the values of paremeters." Based on these two concepts, the probability density function f that gives the probability distribution of $z^u[k](0 \leq k \leq K-1)$, which is an element of z^u, is defined as follows:

Definition 1

$$f(z^u[k])_{\mu^u,\sigma^u} = \begin{cases} \frac{1}{A}Normal(z^u[k], |\mu^u|, \sigma^u) & (z^u[k] \geq 0) \\ \frac{1}{A}Normal(z^u[k], -|\mu^u|, \sigma^u) & (z^u[k] < 0) \end{cases}$$

where μ^u and σ^u are parameters for adjusting the occurrence probability of z^u in P. *Normal* is the probability density function of a normal distribution, that is, $Normal(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$. And A is a normalizing constant defined as follows:

Definition 2

$$A = 2\left((1/2) + \int_0^{|\mu^u|} Normal(x, |\mu^u|, \sigma^u)dx\right)$$

In Fig. 7, functions f when μ^u and σ^u are changed are shown as solid lines. For comparison, $P = \mathcal{N}(0,1)$ is shown as a dotted line. As shown in Fig. 7, it is highly likely that the occurrence probability of $z^u[k]$ obtained from the distribution given by f is low (but not too low) at $P = \mathcal{N}(0,1)$. Moreover, by changing the values of μ^u and σ^u, it is possible to adjust the occurrence probability of $z^u[k]$ in P. To sample $z^u[k]$ from f, Markov Chain Monte Carlo methods can be used. UTG adopts the Metropolis algorithm [23]. Likely-unsupposable dataset LU is generated by inputting $z^u = \{z^u[0], ..., z^u[K-1]\}$ to the decoder of the trained VAE.

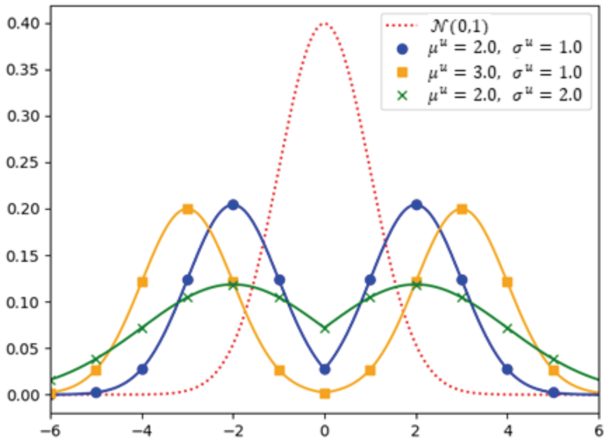


Fig. 7 Probability density functions f of $z^u[k]$

4.3 Implementation on VQ-VAE with PixelCNN

As UTG for image data, implementing UTG by using VQ-VAE with PixelCNN is proposed. By implementing UTG on VQ-VAE with PixelCNN, it is possible to generate unsupposable data with better image quality than that possible with VAE. On the contrary, VQ-VAE with PixelCNN cannot be applied to structured data, so when UTG is applied to structured data, the implementation of UTG with VAE is used. As shown in Section 3.2, in the case of VQ-VAE with PixelCNN, PixelCNN is used to estimate prior distribution P. Specifically, a discrete latent map $\hat{z}$ is acquired on the basis of the categorical distribution output by PixelCNN. Therefore, by manipulating this categorical distribution, $\hat{z^u}$ with a low occurrence probability in P is obtained (Fig. 8). The categorical distribution output by PixelCNN is given as d, which represents the probability that code vector $c[0], ..., c[v], ..., c[V-1]$ contained in the codebook will be selected as $\hat{z_{i,j}}$. Thus, d is expressed as

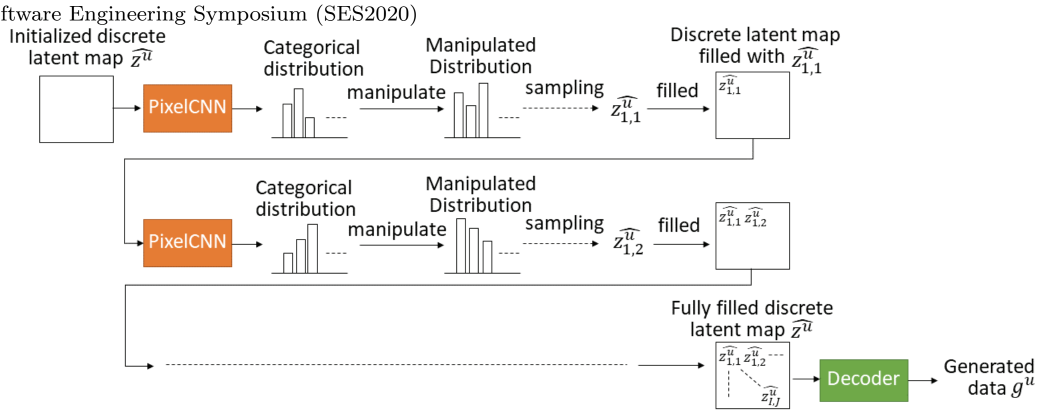


Fig. 8 Implementation of UTG on VQ-VAE with PixelCNN

$d = \{d[0], ..., d[v], ..., d[V-1]\}(\sum_{0\le v\le V-1} d[v] = 1)$. Various possible methods for manipulating d so as to acquire z^u with low occurrence probability in P are available; however, hereafter, the following algorithm 1 is adopted:

Algorithm 1 Algorithm for manipulating categorical distribution d

```
Input: $d, t$
Output: $d^u$

$d^u \leftarrow d$
$sum \leftarrow 0$
for each $v$ in $\{0, ..., V-1\}$ do
    if $d[v] > t$ then
        $diff \leftarrow d[v] - t$
        $sum \leftarrow sum + diff$
        $d^u[v] \leftarrow t$
    end if
end for
for each $v$ in $\{0, ..., V-1\}$ do
    $d^u[v] \leftarrow d^u[v] + (sum/V)$
end for
return $d^u$
```

As for this algorithm, when $d[v](0 \le v \le V-1)$ exceeds threshold t, the value of $d[v]$ is reduced to t. Then, the values of $diff$ for which $d[v]$ exceeds t are totaled and evenly distributed to each $d[v]$. With this algorithm, the code vector that has low probability of being selected from the original d is selected with higher probability. $\hat{z}^u$ is created by using categorical distribution d^u provided in this algorithm, and it is input into the trained decoder to generate likely-unsupposable dataset LU.

5. Experiment

5.1 House Sales Price Dataset

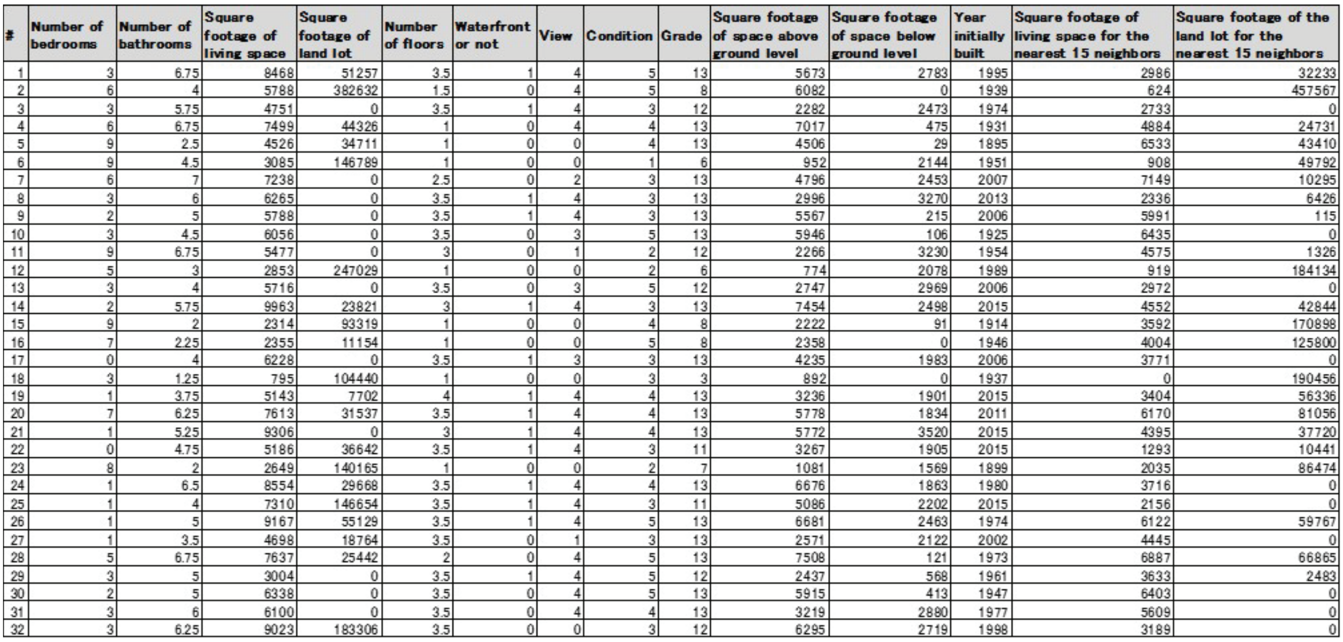
UTG was applied to the dataset [*1] used for the developing a housing-price-forecast model. This dataset consists of 18 attributes (excluding price), and of those attributes, 14 attributes (such as number of bedrooms, number of bathrooms, and size of living room) were selected and taken as existing dataset T. Since this data is not image data but structured data, the VAE implementation described in Section 4.2 was applied. The value of parameter (μ^u, σ^u) was determined by trial and error to be $(5, 5)$. Here, trial and error means the following: the value of (μ^u, σ^u) is tentatively set, and LU generated as a result is referred to. The developer changes the value of (μ^u, σ^u) several times so as to maximizes the number of unsupposable data in generated LU. In that referential way, the value of $(\mu^u, \sigma^u$ is determined. By acquiring 100 values of z^u and inputting them into the decoder, a likely-unsupposable dataset LU consisting of 100 items of g^u was generated. Of the element values of generated data g^u, the values given as integers or categorical values are rounded. Then, for all items of data g^u, whether data with similar characteristics to g^u is included in the existing dataset T and whether g^u is real data were judged manually. If T does not contain data with similar characteristics to g^u and if g^u is real data, g^u is possibly unsupposable data. However, whether the data is actually unsupposable depends on the developer. In this way, data that may be taken as unsupposable data depending on the developer is referred to as *unsupposable-data candidate* hereafter. It was confirmed that at least 32 unsupposable-data candidates (shown in Table 1) were included in the generated LU.

For example, data #1 has 6.75 bathrooms and 3.5 floors, and data which has similar values is not included in existing dataset T. Moreover, other data with more floors (e.g., 3.5 floors) than bathrooms (e.g., 2.5 bathrooms) and data with more bathrooms than floors (e.g., 8 bathrooms and 2.5 floors) is included in T. Data #1 is considered to be real because it seems that the number of bathrooms and floors is better balanced than the other data. Similarly, as for data #2, the living area is about 6,000 square feet, and the average square footage of land lot of 15 neighbors is about 460,000 square feet; however, similar data is not included in T. Data for the square footage of 15 lots of the nearest neighbors is about 9000 square feet with the size of the living area of about 12,000 square feet and data for the average square footage of 15 lots of the nearest neighbors is 870,000 square feet with the size of the living area of about 6,000 square feet, are included in T; therefore, data #2, which has better balance than these existing data, is considered to be real data. From the above results, it can be said data #1 and #2 are unsupposable-data candidates. Similarly, it is affirmed that the other data in Table 1 are real, but they have characteristics that are not included in T.

5.2 MNIST Dataset

UTG was applied with the MNIST dataset (used for developing

[*1] https://www.kaggle.com/harlfoxem/housesalesprediction



#	Number of bedrooms	Number of bathrooms	Square footage of living space	Square footage of land lot	Number of floors	Waterfront or not	View	Condition	Grade	Square footage of space above ground level	Square footage of space below ground level	Year initially built	Square footage of living space for the nearest 15 neighbors	Square footage of the land lot for the nearest 15 neighbors
1	3	6.75	8468	51257	3.5	1	4	5	13	5673	2783	1995	2986	32233
2	6	4	5788	382632	1.5	0	4	5	8	6082	0	1939	624	457567
3	3	5.75	4751	0	3.5	1	4	3	12	2282	2473	1974	2733	0
4	6	6.75	7499	44326	1	0	4	4	13	7017	475	1931	4884	24731
5	9	2.5	4526	34711	1	0	0	4	13	4506	29	1895	6533	43410
6	9	4.5	3085	146789	1	0	0	1	6	952	2144	1951	908	49792
7	6	7	7238	0	2.5	0	2	3	13	4796	2453	2007	7149	10295
8	3	6	6265	0	3.5	1	4	3	13	2996	3270	2013	2336	6426
9	2	5	5788	0	3.5	1	4	3	13	5567	215	2006	5991	115
10	3	4.5	6056	0	3.5	0	3	5	13	5946	106	1925	6435	0
11	9	6.75	5477	0	3	0	1	2	12	2266	3230	1954	4575	1326
12	5	3	2853	247029	1	0	0	2	6	774	2078	1989	919	184134
13	3	4	5716	0	3.5	0	3	5	12	2747	2969	2006	2972	0
14	2	5.75	9963	23821	3	1	4	3	13	7454	2498	2015	4552	42844
15	9	2	2314	93319	1	0	0	4	8	2222	91	1914	3592	170898
16	7	2.25	2355	11154	1	0	0	5	8	2358	0	1946	4004	125800
17	0	4	6228	0	3.5	1	3	3	13	4235	1983	2006	3771	0
18	3	1.25	795	104440	1	0	0	3	3	892	0	1937	0	190456
19	1	3.75	5143	7702	4	1	4	4	13	3236	1901	2015	3404	56336
20	7	6.25	7613	31537	3.5	1	4	4	13	5778	1834	2011	6170	81056
21	1	5.25	9306	0	3	1	4	4	13	5772	3520	2015	4395	37720
22	0	4.75	5186	36642	3.5	1	4	3	11	3267	1905	2015	1293	10441
23	8	2	2649	140165	1	0	0	2	7	1081	1569	1899	2035	86474
24	1	6.5	8554	29668	3.5	1	4	4	13	6676	1863	1980	3716	0
25	1	4	7310	146654	3.5	1	4	3	11	5086	2202	2015	2156	0
26	1	5	9167	55129	3.5	1	4	5	13	6681	2463	1974	6122	59767
27	1	3.5	4698	18764	3.5	0	1	3	13	2571	2122	2002	4445	0
28	5	6.75	7637	25442	2	0	4	5	13	7508	121	1973	6887	66865
29	3	5	3004	0	3.5	1	4	5	12	2437	568	1961	3633	2483
30	2	5	6338	0	3.5	0	4	5	13	5915	413	1947	6403	0
31	3	6	6100	0	3.5	0	4	4	13	3219	2880	1977	5609	0
32	3	6.25	9023	183306	3.5	0	0	3	12	6295	2719	1998	3189	0

Table 1 Unsupposable data in House Sales Price dataset

digit-recognition models) as existing dataset T. Since this dataset contains image data, the implementation of UTG using VQ-VAE with PixelCNN (described in Section 4.3) was applied. Similar to Section 5.1, the value of parameter t was changed by trial and error and finally set to $t = 0.6$. Generated LU (consisting of 100 images) is shown in Fig. 9.

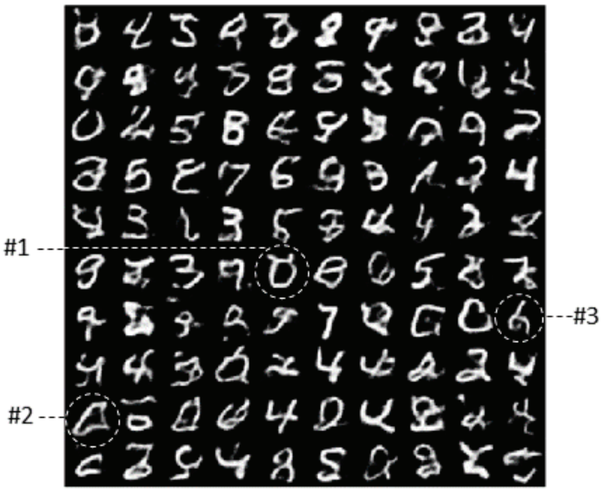


Fig. 9 Example of data generated with MNIST dataset

As for image #1 in Fig. 9, it looks like a zero with short lines like rabbit's ears, and it is affirmed that images with similar characteristics are not included in T. Similarly, it is affirmed that a zero image #2—with a sharp point on its right side—is not included in T. Moreover, image #3 has a point at its center of the circle of the six and the bottom line is faint and disappears. It is affirmed that images with these characteristics are not included in T. Moreover, #1, #2, and #3 are real because they can be recognized as numbers. Therefore, it can be said that these images are unsupposable-data candidates. Other images included in Fig. 9 may also be unsupposable data candidates; however, in this study, only images #1 to #3 were affirmed.

6. Evaluation and Discussion

From the results presented in Section 5, it was confirmed that applying the proposed UTG can generate unsupposable-data candidates. Since unsupposable-data candidates can be unsupposable data depending on the developer, it means that UTG can be useful for generating unsupposable data. By referring to the generated unsupposable data, the developer can recognize unsupposable features and create different unsupposable data having those features. Then, by testing the model by using the unsupposable data generated by UTG and the unsupposable data created by the developer, the developer can confirm the behavior of the model more strictly than hitherto possible.

Two implementations of UTG were shown: VAE (in Section 4.2) and VQ-VAE with PixelCNN (in Section 4.3). If the target data is structured data, it is supposed that the VAE implementation is used. If the target data is image data, the image quality of the generated data is expected to be higher than that of the VAE implementation, so the VQ-VAE-with-PixelCNN implementation is preferred. In addition, VQ-VAE-2 [17] has been proposed as a method of generating data with higher image quality than that possible with VQ-VAE. By implementing the proposed UTG with VQ-VAE-2, it is possible to further improve the quality of generated data.

The concept of UTG is to acquire a latent value with a low occurrence probability in prior distribution P and use it to generate data. That is, the probability density function shown in Definition 1 and the algorithm shown in Algorithm 1 are examples of the method of acquiring a latent value, and it is also possible to acquire a latent value by other methods. For example, it is conceivable to modify Algorithm 1 to create an algorithm by which the values of $diff$ for which $d[v]$ exceeds threshold t are totaled and evenly distributed only to $d[v]$ that do not exceed t. Several conceivable methods were tried, and Algorithm 1 was adopted as the method for obtaining the most unsupposable data as for the MNIST dataset. Different algorithms may be suitable for other datasets.

As stated in Section 1, data can be classified into three types: (1) supposable data, (2) unsupposable data, and (3) unreal data. Dataset of (1) includes existing dataset T and data that can be assumed from T; that is, data with characteristics similar to T. On the contrary, dataset of (3) does not have characteristics similar to T. UTG is based on the assumption that "In the process in which the data of (1) gradually loses the features included in T and eventually changes to the data of (3), it may temporarily becomes the data of (2)." For example, if parameter (μ^u, σ^u) of UTG on VAE is taken as (0, 1), the distribution given by f agrees with P. Accordingly, most of the data included in LU is supposable data. Hereafter, the higher the value of (μ^u, σ^u), the lower the occurrence rate of the acquired latent value in P. As described in Section 3.1, P is the distribution of (unobservable) features of the data included in T. Therefore, data generated from a latent value with high occurrence probability in P has similar characteristics to the data included in T. Conversely, the lower the probability of occurrence of the latent value in P, the less likely it will be that the generated data will have similar characteristics to the data contained in T. That is, if the value of (μ^u, σ^u) is gradually increased, the generated data changes from the data with characteristics similar to T [i.e., data of (1)] to the data with no characteristics of T [i.e., data of (3)]. In a similar manner, with the implementation of UTG with VQ-VAE with PixelCNN, when the value of parameter t is gradually decreased from 1, the generated data changes from (1) to (3).

UTG is based on the assumption that it is possible for data to pass through (2) in the process of changing from (1) to (3). For example, in regard to the MNIST dataset, this assumption is considered to hold. An example of the generated data when parameter t is changed in the UTG implementation with VQ-VAE with PixelCNN is shown in Fig. 10. This figure confirms that as t decreases, the data changes from (1) [supposable] to (2) [unsupposable] to (3) [unreal]. Whether this assumption holds or not depends on the problem domain. Data space of (1) contains data having features that make sense in the problem domain. On the contrary, data space of (3) is an area of data that does not have those features. In the process of changing from (1) to (3), meaningful features are gradually lost. UTG is effective when data with partially lost features that make sense can become unsupposable data. For example, in the case of the MNIST dataset, in the process of changing from (1) to (3), the numerical (visual) features are gradually lost, and a non-numeric image is finally generated. The image that partially lost its numerical features generated is not an image that does not make sense as a digit at all; it is sometimes regarded as an image of a digit that "collapse" by handwriting. In this way, UTG is considered to be effective in problem domains in which if data partially loses its features, it still makes sense.

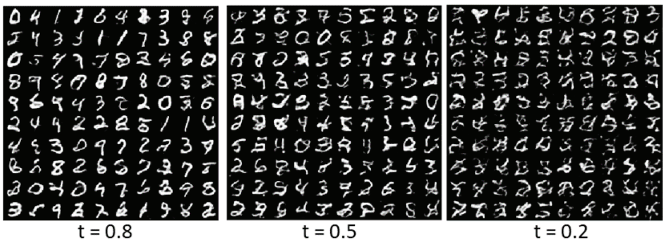


Fig. 10 Change of generated data with changing t

As a typical problem domain in which UTG is not very effective, the image-recognition problem of an object with a complicated occurrence can be considered. For example, living things and vehicles are composed of various combinations of characteristics, and all the characteristics are occurrence-constituent elements of the object. It is therefore highly likely that an image that has partially lost its characteristics becomes unreal data instead of unsupposable data. For example, an airplane with a missing piece of wing is no longer an airplane. From the above considerations, it is considered that the effectiveness of UTG depends on whether the meaning is completely lost when the features that constitute the meaning in the target problem domain are partially lost. Furthermore, even if the data generated by UTG is real data, whether it is unsupposable data depends on the knowledge and experience of the developer.

7. Conclusion

A method called UTG—for generating unsupposable data for developers by utilizing VAE—was proposed. As for UTG, a latent value with a low occurrence probability in the prior distribution of the VAE is obtained. Unsupposable data can be generated by inputting the acquired latent value into the VAE decoder. By referring to the generated data, the developer can recognize new unsupposable features and create another unsupposable data with those features. Then, by testing the model using the unsupposable data generated by UTG and the unsupposable data created by the developer, the developer can confirm the behavior of the model more strictly than hitherto possible. Methods for implementing UTG on VAE and on VQ-VAE with PixelCNN were described. It was also shown that UTG can be useful for generating unsupposable data when applied to the MNIST dataset and the House Sales Price dataset.

As for future work, UTG will be applied to other data sets to confirm its effectiveness. In particular, by applying UTG to image data such as the CIFAR-10 dataset [25], it is hoped to confirm the assumption about its effectiveness discussed in Section 6. It will also be evaluated whether the image quality of the generated data can be further improved by implementing UTG on VQ-VAE-2.

References

[1] G. Katz, C. Barrett, D.L. Dill, K. Julian, and M.J. Kochenderfer: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks, Computer Aided Verification 2017, pp. 97-117 (2017).

[2] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu: Safety Verification of Deep Neural Networks, Computer Aided Verification 2017, Lecture Notes in Computer Science, vol.10426, pp.3-29 (2017).

[3] R. Ehlers: Formal verification of piece-wise linear feed-forward neural networks, Automated Technology for Verification and Analysis

(2017).

[4] H. Tran, P. Musau, D. Manzanas Lopez, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson: Star-Based Reachability Analsysis for Deep Neural Networks, In 23rd International Symposisum on Formal Methods (2019).

[5] N. Sato, H. Kuruma, Y. Nakagawa, and H. Ogawa: Formal Verification of a Decision-tree Ensemble Model and Detection of its Violation Ranges, IEICE Transaction D, Vol.E103-D, No.02, pp.363-378 (2020).

[6] Y. Tian, K. Pei, S. Jana, and B. Ray.: DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars, ICSE'2018 Technical Papers (2018).

[7] K. Pei, Y. Cao, J. Yang, and S. Jana.: DeepXplore: Automated Whitebox Testing of Deep Learning Systems, The 26th ACM Symposium on Operating Systems Principles (2017)

[8] A. Odena and I. Goodfellow: Tensorfuzz: Debugging neural networks with coverage-guided fuzzing, ICML (2019).

[9] M.Y. Liu, T. Breuel, and J. Kautz: Unsupervised Imageto-image Translation Networks, In Adv. NIPS, pp.700-708 (2017).

[10] J.-Y. Zhu, T. Park, P. Isola, A. A. Efros: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, In Proc. International Conference on Computer Vision (ICCV) (2017).

[11] 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. ASE'18 (2018).

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio: Generative adversarial nets, Advances in Neural Information Processing Systems (2014).

[13] M.-Y. Liu and O. Tuzel: Coupled generative adversarial networks, Advances in Neural Information Processing Systems (2016).

[14] D. P. Kingma and M. Welling: Auto-encoding variational bayes, International Conference on Learning Representations (2014).

[15] A.B.L. Larsen, S.K. Sonderby, H. Larochelle, and O. Winther: Autoencoding beyond pixels using a learned similarity metric, International Conference on Machine Learning (2016).

[16] D.J. Rezende, S. Mohamed, and D. Wierstra: Stochastic backpropagation and variational inference in deep latent gaussian models, International Conference on Machine Learning (2014).

[17] A. Razavi, A. v. d. Oord, and O. Vinyals: Generating Diverse High-Fidelity Images with VQ-VAE-2, Advances in Neural Information Processing Systems (NIPS) 32 (2019).

[18] A.v.d. Oord, O. Vinyals, and K. Kavukcuoglu: Neural discrete representation learning, CoRR, abs/1711.00937 (2017).

[19] Z.Q. Zhou and L. Sun: Metamorphic Testing of Driverless Cars, Comm. ACM, vol.62, no.3, pp.61-67 (2019).

[20] T.Y. Chen, S.C. Chung, and S.M. Yiu: Metamorphic Testing - A New Approach for Generating Next Test Cases,HKUST-CS98-01, The Hong Kong University of Science and Technology (1998).

[21] T.Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, Y.H. Tse, and Z.Q. Zhou: Metamorphic Testing: A Review of Challenges and Opportunities, ACM Computing Surveys, vol.51, no.1, Article No.4, pp.1-27 (2018).

[22] I.J. Goodfellow, J. Shelens, and C. Szegedy: Explaining and Harnessing Adversarial Examples, ICRL2015, arXive:1412.6572 (2014).

[23] I. Beichl and F. Sullivan: The Metropolis Algorithm, in Computing in Science & Engineering, vol.2, no.1, pp.65-69 (2000).

[24] Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/. (Accessed on April 28th, 2020)

[25] A. Krizhevsky: The CIFAR-10 dataset, CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/ kriz/cifar.html (Accessed on April 28th, 2020)