Inverted Generator Classifier

Accurate and robust gradient-descent based classifier

Jeongik Cho[1]

Dept. of Computer Science and Engineering[1]

College of Engineering[1]

Konkuk University, Seoul, Korea[1]

jeongik.jo.01@gmail.com[1]

Abstract

In the field of deep learning, a traditional classifier receives input data and passes through hidden layers to output predicted labels. Conditional generators such as Conditional VAE [1] and Conditional GAN [2] receive latent vector and condition vector and generate data with the desired conditions.

In this paper, I propose an Inverted Generator Classifier that uses conditional generators to find a pair of condition vectors and latent vectors that generate the data closest to the input data, and predict the label of the input data. Inverted Generator Classifier uses a trained conditional generator as it is. The inverted generator classifier repeatedly performs gradient descent by taking the latent vector for each condition as a variable and the model parameter as a constant to find the data closest to the input data. Then, among the data generated for each condition, the condition vector of the data closest to the input data becomes the predicted label.

Inverted Generator Classifier is slow to predict because prediction is based on gradient descent, but has high accuracy and is very robust against adversarial attacks [3] such as noise. It is also not subject to gradient-descent based white-box attacks like FGSM [4].

Abbreviations

Inverted Generator Classifier: IGC

1. Inverted Generator Classifier

1.1 Training

IGC uses trained conditional generators such as Conditional VAE and Conditional GAN as models. For conditional VAE, a decoder is used, and for conditional GAN, a generator is used as a model for IGC. No additional training is required after training the conditional generator.

1.2 Prediction

First, IGC finds a pair of condition vectors and latent vectors that generate data closest to input data through the latent space search. Then, among the data generated for each condition, the condition vector of the data closest to the input data becomes the predicted label.

The latent space search is to perform multiple gradient descents taking the latent vector for each condition as a variable, the model parameter as a constant, and using two losses: data difference loss and latent restriction loss. Through this, a pair of condition vectors and latent vectors that generate data close to the input data can be found.

The data difference loss is the loss to find the latent vector that can generate the data closest to the input data for each condition. The latent restriction loss is a loss for the latent vector not to search beyond the latent space too much.

The loss for IGC to perform latent space search is as follows.

$$L = L_{DD} + \lambda_{LR} L_{LR}$$

$$L_{DD} = \sum_{(cnd,ltn)\in S_{in\_vec}} dif(G(cnd, ltn), d)$$

$$L_{LR} = \sum_{(cnd,ltn)\in S_{in\_vec}} average\Big(abs\big(z\_score(ltn)\big)\Big)$$

L is the loss for IGC to perform latent space search through gradient descent. L_{DD} is data difference loss, and L_{LR} is latent restriction loss. λ_{LR} is the weight of latent restriction loss.

S_{in_vec} is a set of pairs having a cnd (condition vector) and a ltn (latent vector). S_{in_vec} has cnd corresponding to each class as many as the number of classes. For example, if there are 10 classes, S_{in_vec} has 10 pairs with different condition vectors. G is a trained conditional generator. $G(cnd, ltn)$ is one data generated by G by receiving cnd and ltn. d is one input data. dif is a function that measures the difference between two data.

z_score is a function that calculates the z score of each element of the input vector based on the distribution of latent vectors used when training G. For example, if G was trained by 3 dimension latent vector, each element of which follows a Gaussian distribution (mean=0, standard deviation=1), $z\_score([1,2,-3])$ is $[1,2,-3]$.

abs is a function that converts each element of the input vector to an absolute value. $average$ is a function to find the average of each element of the input vector.

To reduce L, gradient descent is performed by taking the latent vector for each condition as variables and the model parameters as constants. If gradient descent is repeatedly performed a certain number of times, the latent space search ends.

Then, the difference between the data generated for each condition and the input data is measured using the dif function, and the condition with the smallest difference is determined as the predicted label.

$$(predicted\ label, latent\ vector) = \underset{(cnd,ltn)\in S_{in\_vec}}{\arg\min}\ dif(G(cnd, ltn), d)$$

Label	Condition Vector				Latent vector		
num 0	1 (untrainable)	0 (untrainable)	0 (untrainable)	…	0.3 (trainable)	-1.0 (trainable)	…
num 1	0 (untrainable)	1 (untrainable)	0 (untrainable)	…	-0.2 (trainable)	0.1 (trainable)	…
num 2	0 (untrainable)	0 (untrainable)	1 (untrainable)	…	0.7 (trainable)	-0.3 (trainable)	…
…	…	…	…	…	…	…	…

Fig.1 Example of input vectors of IGC

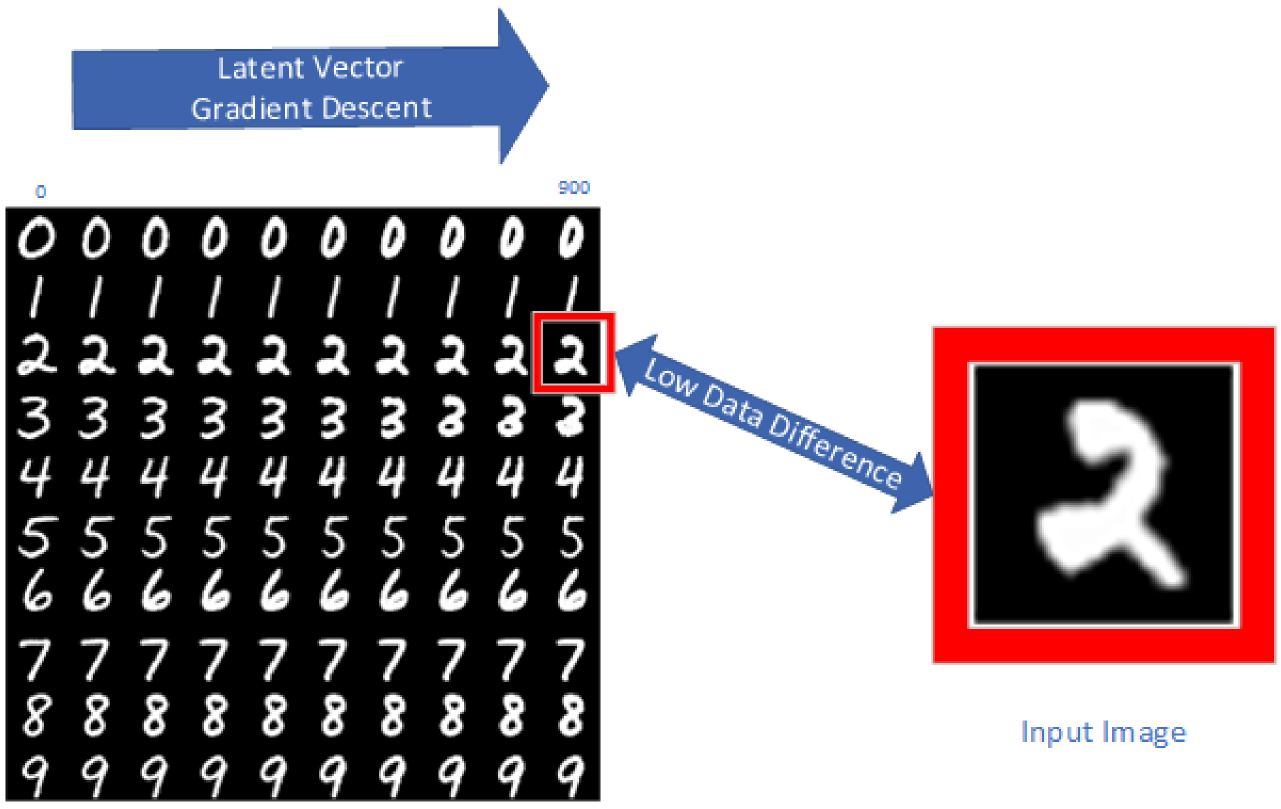

Fig.2 Prediction process of IGC

Fig. 1 is an example of an input vectors of IGC. The condition vector, which is an untrainable variable, does not change when performing gradient descent. However, the latent vector, which is a trainable variable, changes with every gradient descent.

Initially, the latent vector is initialized with the average of the latent vector distribution used during generator training. That is, at first, all latent vector for each condition are the same. Later, the latent vector changes to generate an image close to the input image.

The leftmost column in Fig. 2 is data generated for each condition before performing gradient descent, and the rightmost column is after gradient descent is performed 900 times. After performing a gradient descent to some extent, the input condition vector to generate data with the closest distance to the input image be the predicted label of the IGC.

1.3 Out of class

Traditional classifier cannot distinguish data that does not belong to any class. For example, in the case of a classifier that classifies the numbers 0 to 9, the classifier will predict the class as one of the numbers 0 to 9 even if noise is input instead of numbers. However, IGC can measure the degree of Out Of Class for input data.

$$ooc = \min_{(cnd,ltn)\in S_{in\_vec}} dif(G(cnd, ltn), d)$$

ooc is degree of Out Of Class. IGC can classify input data as out of class when ooc is more than a specific value.

1.4 Multi-label classification

In multi-class classification with one label, IGC can predict the label of one data by creating pairs of condition vector and latent vector as many as the number of classes of the label. However, in the case of multi-label classification, the time required for prediction may be too long because there are so many possible combinations of condition vectors.

Instead, the IGC can shorten the time for prediction by repeating prediction for each label. That is, when predicting a label, the condition vector for the label is set as an untrainable variable, and the condition vectors for the remaining labels and latent vector are set as trainable variables to perform a latent space search. This prediction must be repeated as many as the number of labels.

2. IGC Experiment

Tensorflow 2.1 without compile option and rtx2080ti was used for the experiment. In this experiment, I used the MNIST handwriting number dataset [5] (60000 images for training, 10000 images for test, 28x28x1 resolution). I used conditional activation GAN [6] with LSGAN [7] adversarial loss to train conditional generator. The generator receives a 10-dimensional condition vector and a 256-dimensional latent vector. All elements of the latent vector used in training follow the Gaussian distribution with mean = 0 and standard deviation = 1. The average FID [8] for each condition of the generator after training was measured to be 2.0. Since the MNIST dataset has one channel and their resolution is too low for the inception network, the width, height, and channel are tripled for the FID evaluation (84 × 84 × 3).

For prediction of IGC, gradient descent was performed 100 times for each image, and Adam optimizer [9] (learning rate = 0.001, beta1=0.9, beta2 = 0.999) was used. The latent restriction loss (λ_{LR}) is 0.03 and the dif function is mean absolute error. All latent vectors are initialized to 0. 1000 data randomly selected from the MNIST test dataset were used for the prediction evaluation.

First, I tested the change in accuracy according to the change in the intensity of random Gaussian noise to test images. The original image was normalized from -0.5 to 0.5, and Gaussian noise with an average of 0 and a standard deviation of 1 was multiplied by sigma σ and added to the original image, and clipped -0.5~0.5 to keep the image stay within range.

$$noised\ image = original\ image + \sigma * noise$$

Test Data size	1000	1000	1000
Sigma	0	0.2	0.4
Accuracy (%)	95.1	94.7	91.2
Time(sec)	2526	2535	2586

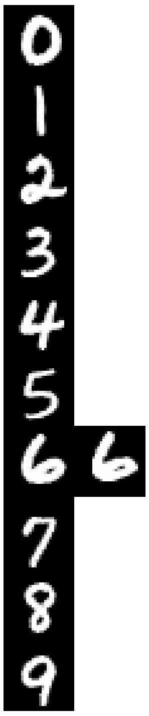
Fig.3 Correct case of IGC predict. Without noise. Number 6 in right side is the input image.

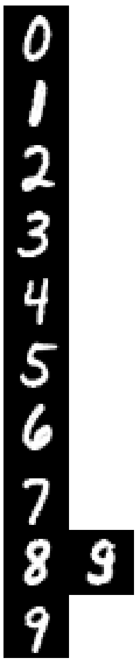
Fig.4 Incorrect case of IGC prediction. Number 9 in right side is the input image but IGC predicted number 8.

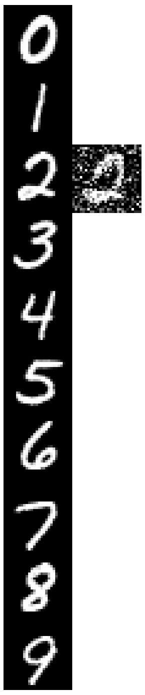
Fig.5 Correct case of IGC predict. sigma=0.4. Noised number 2 in right side is the input image.

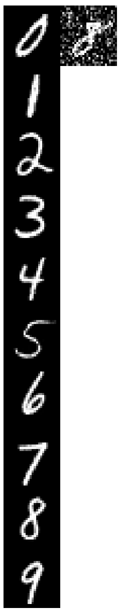

Fig.6 Incorrect case of IGC prediction. Noised number 8 in right side is the input image but IGC predicted number 0.

Fig.7 Out of class example. ooc=0.31031805

Fig.8 Not out of class example. ooc=0.02960496

Fig. 7 and 8 show ooc value for the input image. For out of class data that does not belong to any class, it has a large ooc value, but for data belonging to a specific class, it has a low ooc value.

3. Parallel Inverted Generator Classifier

Gradient descent-based searches are always likely to converge to local optima, not global optima. Likewise, in the latent space search, there is a possibility that the latent vector falls into the local optima instead of the global optima. To increase the probability that the IGC finds a latent vector falling into the global optima, or a little better local optima, Parallel IGC can be used. IGC searched one latent vector per condition, but Parallel IGC searched multiple latent vectors per condition to perform the latent space search. Also, latent vectors corresponding to each condition of IGC are initialized with the average of the latent vectors used in conditional GAN training, but parallel IGC latent vectors are initialized randomly with the one average latent vector to find different local optima.

Label	Condition Vector				Latent Vector		
num 0	1 (untrainable)	0 (untrainable)	0 (untrainable)	…	0.0 (trainable)	0.0 (trainable)	…
num 0	1 (untrainable)	0 (untrainable)	0 (untrainable)	…	0.7 (trainable)	-0.2 (trainable)	…
num 0	1 (untrainable)	0 (untrainable)	0 (untrainable)	…	-0.6 (trainable)	0.1 (trainable)	…
num 1	0 (untrainable)	1 (untrainable)	0 (untrainable)	…	0.0 (trainable)	0.0 (trainable)	…
num 1	0 (untrainable)	1 (untrainable)	0 (untrainable)	…	0.7 (trainable)	-0.8 (trainable)	…
…	…	…	…	…	…	…	…

Fig.7 Example of initialized input vectors of Parallel IGC

4. Parallel IGC Experiments

All experimental conditions are the same as 2.IGC experiments. The latent vector size per condition is 5.

Test data size	1000	1000	1000
Sigma	0	0.2	0.4
Accuracy (%)	95.7	95.7	92.3
Time (sec)	4201	4205	4208

Fig.8 Parallel IGC prediction accuracy

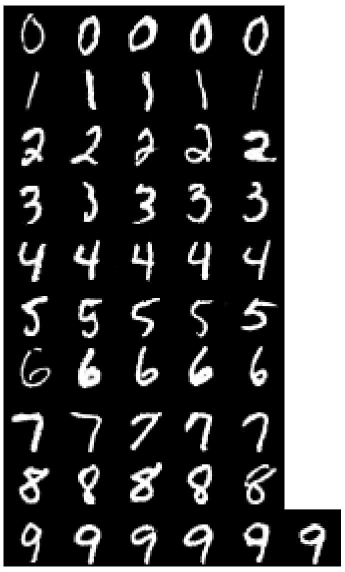

Fig.3 Correct case of Parallel IGC predict. Without noise. Number 9 in right side is the input image.

5. Conclusion

Inverted Generator Classifier is slow when predicting because it predicts based on gradient descent, but accuracy is high and very robust against adversarial attacks such as noise. Also, it is impossible to apply the existing white-box adversarial attack such as FGSM, so it is safe in terms of security.

6. References

[1] Kihyuk Sohn, Honglak Lee, Xinchen Yan

Learning Structured Output Representation using Deep Conditional Generative Models

https://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models

[2] Mehdi Mirza, Simon Osindero

"Conditional Generative Adversarial Nets", arXiv preprint arXiv:1411.1784, 2014.

https://arxiv.org/abs/1411.1784 (accessed 16 February 2020)

[3] Xiaoyong Yuan, Pan He, Qile Zhu, Xiaolin Li

Adversarial Examples: Attacks and Defenses for Deep Learning

https://arxiv.org/abs/1712.07107

[4] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy

Explaining and Harnessing Adversarial Examples

https://arxiv.org/abs/1412.6572

[dataset] [5] Yann LeCun, Corinna Cortes, Christopher J.C. Burges

THE MNIST DATABASE of handwritten digits

http://yann.lecun.com/exdb/mnist/

[6] JeongIk Cho, Kyoungro Yoon

Conditional Activation GAN: Improved Auxiliary Classifier GAN

http://vixra.org/abs/1912.0204

[7] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, Stephen Paul Smolley

Least Squares Generative Adversarial Networks

The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2794-2802

https://ieeexplore.ieee.org/document/8237566

[8] Heusel, Martin and Ramsauer, Hubert and Unterthiner, Thomas and Nessler, Bernhard and Hochreiter, Sepp

GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium Advances in Neural Information Processing Systems 30 (NIPS), 2017, pp. 6626-6637 https://papers.nips.cc/paper/7240-gans-trained-by-a-two-time-scale-update-rule-converge-to-a-local-nash-equilibrium

[9] Diederik P. Kingma, Jimmy Ba

Adam: A Method for Stochastic Optimization

https://arxiv.org/abs/1412.6980