Inverted Conditional Generator Classifier

Slow but accurate and robust gradient-descent based prediction 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

Traditional deep neural network classifier receives input data and passes through hidden layers to output predicted labels.

Conditional generator such as Conditional VAE [1] or Conditional GAN [2] receives latent vector and condition vector, and generates data with the desired conditions.

In this paper, I propose an Inverted Conditional Generator Classifier that uses conditional generators to find a pair of condition vector and latent vector that can generate the data closest to the input data, and predict the label of the input data. The inverted Conditional Generator Classifier uses a trained conditional generator as it is.

The inverted conditional 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 Conditional 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. In addition, the Inverted Conditional Generator Classifier can measure the degree of out-of-class through the difference between the generated nearest data and input data.

Abbreviations

Inverted Conditional Generator Classifier: ICGC

Deep Neural Network: DNN

1. Introduction

Traditional deep neural network classifiers can be very sensitive to small changes in input data [4]. Using this instability of the DNN classifier, many successful adversarial attack methods [+] to deceive the classifier with small data changes have been studied.

The conditional generator is a generator that receives condition vector and latent vector, and generates data with the desired conditions. A decoder of conditional VAE or a generator of conditional GAN, or other conditional generative models can be a conditional generator.

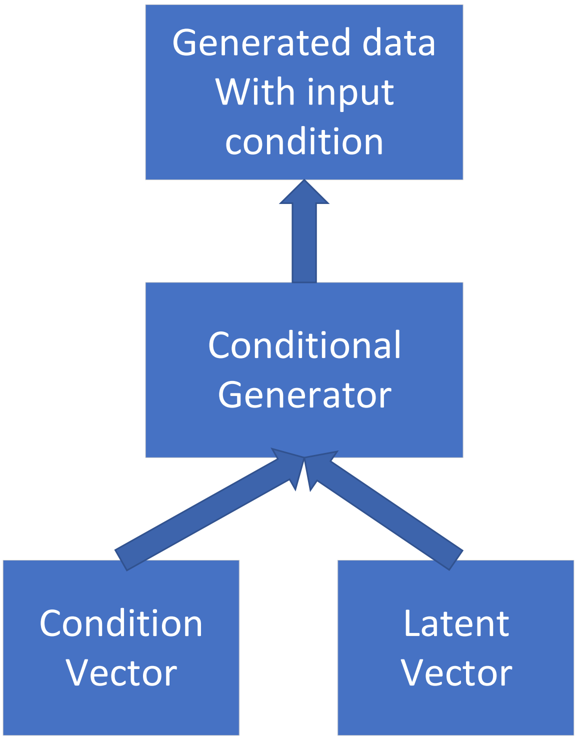

Fig.1 Conditional Generator

In this paper, I propose a new classifier called ICGC that performs gradient descent-based prediction using a conditional generator, rather than a traditional deep neural network classifier that outputs a predicted label through a hidden layer. ICGC uses conditional generator to find the pair of condition vector and latent vector that can generate the data closest to the input data through gradient descent, and outputs the condition vector of the data as a predicted label.

Since ICGC classifies the data by generating the data closest to the input data, it is not sensitive to small changes like the traditional DNN classifier, so it is very resistant to adversarial attacks.

The traditional DNN classifier cannot classify the input data as out-of-class even if it belongs to out-of-class. For example, in the case of a DNN classifier that classifies the numbers 0 to 9, when a noise image is input, it cannot be predicted as out-of-class. However, since ICGC generates the data closest to the input data among the data that the conditional generator can generate, the degree of out-of-class can be measured through the difference between the generated data and the input data. Using this, ICGC can classify the input data as out-of-class when the degree of out-of-class is more than a certain value.

2. Inverted Conditional Generator Classifier

2.1 Training

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

2.2 Prediction

First, ICGC finds a pair of condition vectors and latent vectors that generate data closest to input data through a 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 to prevent the latent vector from searching too far from the latent space used for conditional generator training.

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

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

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

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

L is the loss for ICGC 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 a pair of cnd corresponding to each class and ltn corresponding to the cnd as many as the number of classes. For example, if there are 10 classes, S_{in_vec} has 10 (cnd, ltn) pairs. G is a trained conditional generator. $G(cnd, ltn)$ is one data generated by G by receiving cnd and ltn. in_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 vector used when training G. For example, when G is trained using a latent vector that follows a Gaussian distribution with mean 0 and 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.

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

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.2 Example of input vectors of ICGC

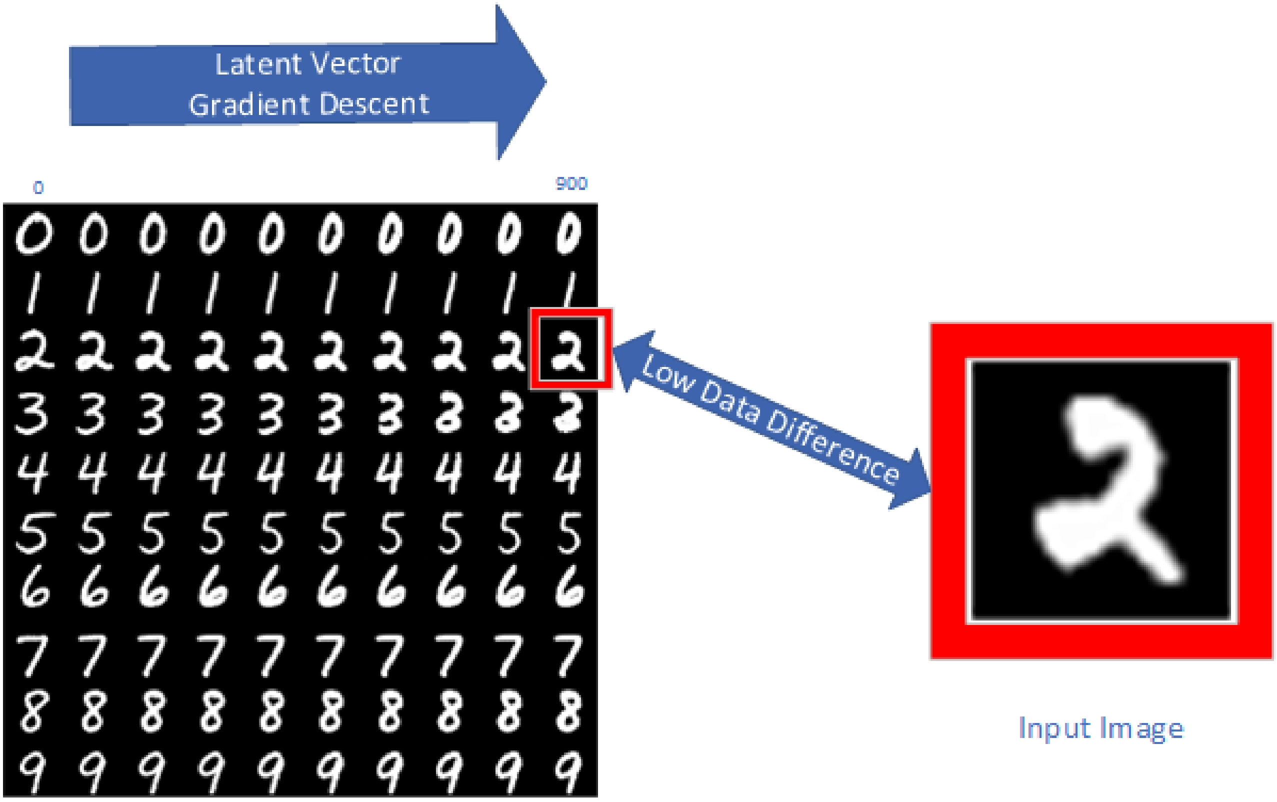

Fig.3 Prediction process of ICGC

Fig.2 is an example of an input vectors of ICGC. 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.

Fig.3 shows the process of ICGC prediction. Initially, all 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.3 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 ICGC.

2.3 Out-of-class

Traditional DNN 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, ICGC can measure the degree of out-of-class for input data.

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

ooc is the degree of out-of-class. ICGC can classify input data as out-of-class when ooc is more than a specific value.

2.4 Multi-label classification

In multi-class classification with one label, ICGC 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 ICGC can shorten the time for prediction by repeating prediction for each label. That is, when performing prediction on one label, the condition vector for the label to be predicted is set as an untrainable variable, and the condition vectors for the remaining labels and latent vector are set as trainable variables to perform latent space search. This prediction must be repeated as many as the number of labels.

2.5 Parallel ICGC

Gradient descent-based search always has the potential to converge to local optima, not global optima. Likewise, there is a possibility that during the latent space search by ICGC, the latent vector falls into the local optima, not the global optima.

To increase the probability that the ICGC finds a latent vector falling into the global optima, or even a little better local optima, Parallel ICGC can be used. ICGC searched one latent vector per condition, but Parallel ICGC searched multiple latent vectors per condition to perform a latent space search. In addition, a latent vector corresponding to each condition of ICGC is initialized with the average of latent vectors used in conditional generator training, but parallel ICGC latent vectors are randomly initialized with the latent vector of ICGC to find different local optima.

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

Fig.4 Example of initialized input vectors of Parallel ICGC

3. 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).

3.1 Training

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).

3.2 ICGC evaluation

For prediction of ICGC, 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. 1000 data randomly selected from the MNIST test dataset were used for the prediction evaluation.

3.2.1 FGSM test

To show that ICGC is resistant to adversarial attacks, I experimented with FGSM attack [10]. FGSM noise is generated in the direction that the loss L for the label predicted by ICGC for the input data increases and the loss L for the other labels decreases.

$$\begin{aligned} &FGSM\ loss \\ &= L_{DD} + \lambda_{LR} L_{LR} \\ &- 2 \\ &\times \left(dif(G(p\_cnd, p\_ltn), in\_d) \right. \\ &\left. + \lambda_{LR} average\left(abs\left(z\_score(p\_ltn) \right) \right) \right) \end{aligned}$$

$$FGSM\ noise = -sgn\left(\frac{\Delta FGSM\ loss}{\Delta in\_d}\right) \div 2$$

p_cnd and p_ltn are condition vectors and latent vectors for generating data closest to input data, respectively. sgn is the sign function.

The original image was normalized to -0.5 to 0.5, FGSM noise was multiplied by sigma, added to the original image, and clipped to maintain the range of -0.5 to 0.5.

$$noised\ image = clip(original\ image + \sigma * FGSM\ noise)$$

Latent vector size / Condition vector size	1	1	1	10	10	10
Sigma	0	0.2	0.4	0	0.2	0.4
Accuracy(%)	95.8	93.3	88.7	96.1	95.5	93.7

Fig.6 Accuracy by latent vector per condition vector, and sigma

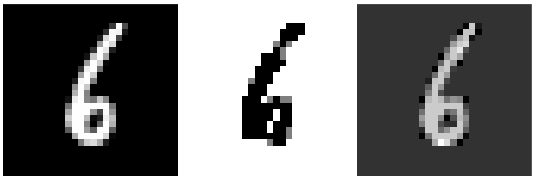

Fig.7 original image, FGSM noise, noised image in turn. $\sigma = 0.4$

Fig.8 Generated images to predict noised image of Fig.7. $ltn/cnd = 1$

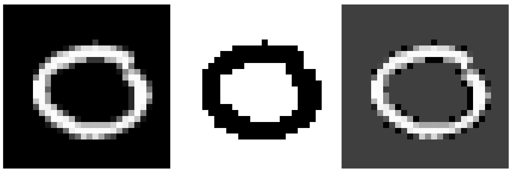

Fig.9 original image, FGSM noise, noised image in turn. $\sigma = 0.4$

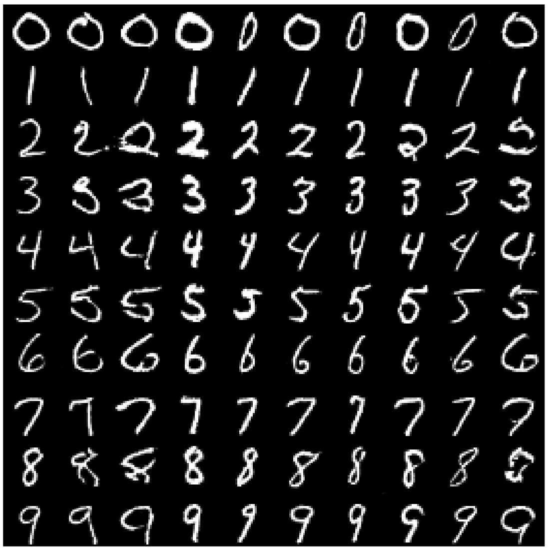

Fig.10 Generated images to predict noised image of Fig.9. $ltn/cnd = 10$

The FGSM noise of the DNN classifier is generally unrecognizable noise, but the ICGC FGSM noise is very similar to the inversion of the input image. In addition, even at high sigma values, the accuracy hardly decreases.

3.2.2 Gaussian noise test

To show that ICGC is robust against any type of adversarial attack, I experimented with a Gaussian noise attack. 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.

$$noised\ image =$$
$$clip(original\ image + \sigma * gaussian\_noise)$$

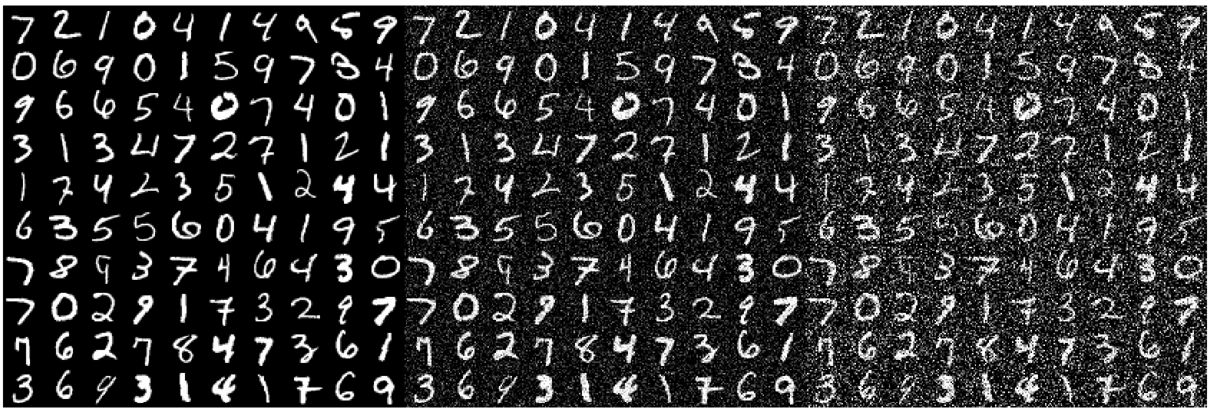

Fig.11 MNIST images with $\sigma = 0.0$, $\sigma = 0.2$, $\sigma = 0.4$ in turn

Latent vector size / Condition vector size	1	1	1	10	10	10
Sigma	0.0	0.2	0.4	0.0	0.2	0.4
Accuracy(%)	95.1	94.7	91.2	96.1	96.0	93.4
Time(sec)	2526	2535	2586	6393	6426	6484

Fig.12 Accuracy by latent vector per condition vector, and sigma

Fig.13 Correct case of ICGC prediction. $\sigma = 0.0$. Number 6 on the right side is the input image.

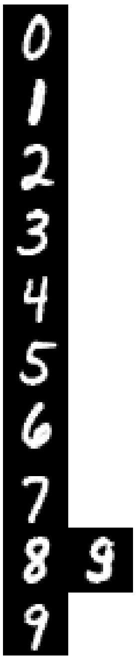

Fig.14 Incorrect case of ICGC prediction. $\sigma = 0.0$. Number 9 on the right side is the input image but ICGC predicted number 8.

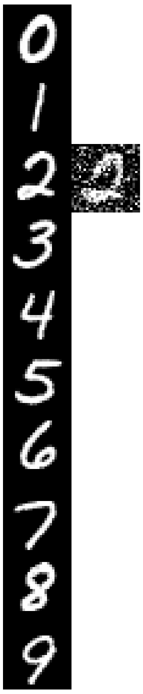

Fig.15 Correct case of ICGC prediction. $\sigma = 0.4$. Noised number 2 on the right side is the input image.

Fig.16 Incorrect case of ICGC prediction. $\sigma = 0.4$. Noised number 8 on the right side is the input image but ICGC predicted number 0.

Fig.17 Correct case of Parallel ICGC predict. $\sigma = 0.0$. Number 9 on the right side is the input image.

The table in Fig.12 shows the difference in accuracy between ICGC and parallel ICGC according to the degree of noise. In particular, in the case of parallel ICGC using 10 latent vectors per condition vector, there is almost no difference in accuracy between $\sigma = 0.0$ and $\sigma = 0.2$.

3.2.3 Out-of-class test

Fig.18 Out-of-class example. ooc=0.31031805

Fig.19 Not out-of-class example. ooc=0.02960496

Fig.18 and 19 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.

4. Conclusion

ICGC is slow when predicting because it predicts based on gradient descent, but accuracy is high and very robust against adversarial attacks.

++DNN classifier compare

++ooc supplement

++multi-label test

++black box test

5. References

++adversarial attack 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] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus

Intriguing properties of neural networks https://arxiv.org/abs/1312.6199

[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

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

Explaining and Harnessing Adversarial Examples

https://arxiv.org/abs/1412.6572