Wasserstein Generative Adversarial Imputation Nets with gradient penalty

Y.Liang
College of Control Science and Engineering
Zhejiang University
Hangzhou, CN 310027
liangyihao.academic@gmail.com

Abstract

We propose an improving method for imputing missing data by adapting the Generative Adversarial Imputation Nets (GAIN) framework based on Wasserstein GAN with gradient penalty. Accordingly, we call our method Wasserstein Generative Adversarial Imputation Nets with gradient penalty (WGAIN_gp). We explain why it is more suitable and stable in high-dimensional data imputation. The method was tested on different datasets, where the values are missing completely at random (MCAR) , and the result shows WGAIN_gp has good performance, compensates for the lack of hint mechanism, even if the number of training epochs is only one-fifth of GAIN's.

1 Limitations of GAIN

1.1 Hint mechanism

The hint mechanism is one of the most innovative points of this work [1], and that's where Nabeel suggested me paying attention to. In J.Yoon's implementation [1], the hint is created by removing ones of the mask vector $\mathbf{M}$ with a probability of 0.9 instead of 0.5. The paper define:

$$\mathbf{H} = \mathbf{B} \odot \mathbf{M} + 0.5(\mathbf{1} - \mathbf{B})$$

Table 1: The relationship between the hint value and the data

$\mathbf{M}$	$\mathbf{B}$	$\mathbf{H}$	Data
1	1	1	Known oringinal data
0	1	0	Known imputed data
1	0	0.5	Unknown
0	0	0.5	Unknown

Table 1 shows the mapping of hint values to data types in theory. However, in this implementation the hint mechanism maps $H_i = 1 \Rightarrow M_i = 1$, which means the known original data, and maps $H_i = 0$ to unknown data. This means the hint is only useful for the known original values insread of the missing values, and the imputed variables isn't provieded with a hint.

I therefore conclude that:

1. It made the best performance in practice when hint rate is 0.9.
2. 0.5 is not used in the implementation.

[1]https://github.com/jsyoon0823/GAIN

3. Hint mechanism does not have much influence where the datasets are MCAR, although the paper showed that the hint had a gain from 4.1% to 14.6% in RMSE.

1.2 Limitations of GAN

I focused my attention on GAN itself to understand the limitations of Generative Adversarial Nets(GAN) [2] and the solutions, and try to apply available improvements to GAIN. Since proposed by Ian Goodfellow in 2014, GAN has suffered from difficulties in training, loss of generators and discriminators that do not indicate the training process, and lack of diversity in generated samples. When training GAN, the generator and discriminator are in an adversarial state, either side is too strong and will crush the other, and the adversarial balance is broken, the training will fail.

Much work like LSGAN [3], SAGAN [4] and CGAN [5] have tried to solve it, but with unsatisfactory results. One of the most famous improvements, DCGAN [6] relies on experimental enumeration of discriminator and generator architectures to eventually find a better set of network architecture settings, but it actually treats the symptoms but not the root cause, and does not solve the problem completely.

1.2.1 The loss of discriminator

The better the discriminator, the more severe the generator gradient vanishing.

The discriminator in the original GAN has to minimize the loss function as Eq.1, classifying the real samples as positive cases and the generated samples as negative cases as far as possible.

$$V(G,D) = -\mathbb{E}_{x\sim P_r}[\log D(x)] - \mathbb{E}_{x\sim P_g}[\log(1-D(x))] \tag{1}$$

where P_r is the true sample distribution and P_g is the sample distribution generated by the generator. For the generator, Goodfellow proposes a loss function, and an improved loss function, as shown in Eqs.2, 3, respectively

$$C(G) = \mathbb{E}_{x\sim P_g}[\log(1-D(x))] \tag{2}$$

$$C(G) = \mathbb{E}_{x\sim P_g}[-\log D(x)] \tag{3}$$

From Eq.1, we can obtain the optimal discriminator D when the generator G has fixed parameters. For a specific sample x, which may come from either the real or the generative distribution, the optimal discriminator is:

$$D^*(x) = \frac{P_r(x)}{P_r(x)+P_g(x)} \tag{4}$$

However, there is a trick in GAN's training, that is, don't train the discriminator too well, otherwise the generator will not learn at all in the experiment (the loss can't be lowered), in order to explore the reason behind, we can see what the loss function of the generator becomes in the extreme case - when the discriminator is optimal. Adding a term to Eq.2 that does not depend on the generator makes it:

$$\mathbb{E}_{x\sim P_r}[\log D(x)] + \mathbb{E}_{x\sim P_g}[\log(1-D(x))]$$

Minimizing this loss function is equivalent to minimizing Eq.2,which is the negative of the discriminant loss function. Substituting into the optimal discriminator Eq.4:

$$\mathbb{E}_{x\sim P_r} \log \frac{P_r(x)}{\frac{1}{2}[P_r(x)+P_g(x)]} + \mathbb{E}_{x\sim P_g} \log \frac{P_g(x)}{\frac{1}{2}[P_r(x)+P_g(x)]} - 2\log 2 \tag{5}$$

After introducing the KL divergence and JS divergence, Eq.5 can be written as

$$2JS(P_r \| P_g) - 2\log 2 \tag{6}$$

We can conclude that we can get the optimal discriminator based on the discriminator loss defined by the original GAN; and under the optimal discriminator, we can equivalently transform the generator loss defined by the original GAN to minimize the JS divergence between the real distribution P_r and the generated distribution P_g . The more we train the discriminator, the closer it will be to the optimal, and the more minimizing the loss of generator will be approximated by minimizing the JS divergence between P_r and P_g.

We hope that if the JS divergence between two distributions is smaller, the closer they are to each other, we can optimize the JS divergence so that we can P_r "pull towards" P_g and eventually make it look fake. This hope is valid when the two distributions overlap, but if the two distributions have no overlap at all, or if their overlap is negligible, their JS divergence is also $\log 2$! The proof is in A.1.

This means that for the gradient descent method the gradient is 0. In this case, for the optimal discriminator, the generator will not get any gradient; even for the near-optimal discriminator, the generator will face the problem of gradient vanishing. And when the support with is a low-dimensional manifold in a high-dimensional space, the probability, that the measure of the the overlap between P_r and P_g is 0, is 1.

Notice that in GAIN, the authors use the same loss function of discriminator as Eq.7 shows. The generator is randomly initialized at the beginning, so it is almost impossible for the distribution of $\mathbf{x}, \mathbf{h}$ between missing data and observed data to be related, which leads to the gradient vanishing problem.

$$D(\mathbf{x}, \mathbf{h})_i = \frac{p\left(\mathbf{x}, \mathbf{h}, m_i = 1\right)}{p\left(\mathbf{x}, \mathbf{h}, m_i = 0\right) + p\left(\mathbf{x}, \mathbf{h}, m_i = 1\right)} \tag{7}$$

If the discriminator is trained too well, the generator gradient disappears and the generator loss does not drop; if the discriminator is not trained well, the generator gradient is not accurate. Only the discriminator training is either not too bad or too good, but this balance is difficult to get, so the GAN is so difficult to train.

1.2.2 The loss of generator

Minimizing the improved generator loss function(Eq.3) would be equivalent to minimizing an unreasonable distance measure, leading to two problems: being gradient instability, and lack of collapse mode, i.e., diversity.

We can obtain an equivalent representation of Eq.3 in terms of JS divergence and KL divergence, as Eq.8. The proof is in A.2.

$$KL\left(P_g \| P_r\right) - 2JS\left(P_r \| P_g\right) \tag{8}$$

There are two serious problems with this equivalence minimization objective.

First, it simultaneously minimizes the KL divergence of the generated distribution and the real distribution, yet maximizes the JS divergence of both, one to bring them closer together, but one to push them farther apart! This is intuitively absurd and numerically leads to unstable gradients, which is the fault of that latter JS divergence term.

Second, there is also a problem with the first normal KL divergence. Because KL divergence is not a symmetric measure, There is a difference between $KL\left(P_g \| P_r\right)$ and $KL\left(P_r \| P_g\right)$.

- When $P_g(x) \rightarrow 0$ and $P_r(x) \rightarrow 1$, $P_g(x) \log \frac{P_g(x)}{P_r(x)} \rightarrow 0$, the contribution to $KL\left(P_g \| P_r\right)$ converges to 0.
- When $P_g(x) \rightarrow 1$ and $P_r(x) \rightarrow 0$, $P_g(x) \log \frac{P_g(x)}{P_r(x)} \rightarrow +\infty$, the contribution to $KL\left(P_g \| P_r\right)$ converges to positive infinity.

In other words, $KL\left(P_g \| P_r\right)$ has different penalties for the above two errors, the first error corresponds to "the generator failed to generate a real sample" with a small penalty, and the second error corresponds to "the generator generated an unreal sample" with a large penalty. The first error corresponds to the lack of diversity, and the second error corresponds to the lack of accuracy.

Therefore, the generator would rather generate more duplicate but "safe" samples than generate diverse samples, because then the second error will happen accidentally. This phenomenon is often named as collapse mode.

2 Improvement of GAIN

Wasserstein Generative Adversarial Nets (WGAN) [7] essentially solves this problem by using a better way to measure the gap between the generated data and the real data, and using this way to bring them closer together. This makes GAN less sensitive to "adversarial balance".

2.1 WGAN

Wasserstein distance is also known as Earth-Mover (EM) distance:

$$W\left(P_r, P_g\right)=\frac{1}{K} \sup _{\|f\|_L \leq K} \mathbb{E}_{x \sim P_r}[f(x)]-\mathbb{E}_{x \sim P_g}[f(x)]$$

The superiority of Wasserstein distance over KL divergence and JS divergence is that even if the two distributions do not overlap, the Wasserstein distance can still reflect their proximity. WGAN replaces JS divergence with Wasserstein distance and accomplishes both stable training and process metrics problems.

Here we do not elaborate on the nature, the proof of W, we only focus on the improvement and how to apply it to GAIN.

WGAN has changed only four things compared to the first form of the original GAN.

- Removes the sigmoid from the last layer of the discriminator.
- The loss of the generator and discriminator is not taken as log.
- Truncate the absolute values of the discriminator parameters to no more than a fixed constant c after each update, which is named as weight clipping.
- Do not use momentum-based optimization algorithms (Momentum and Adam), RMSProp and SGD are also recommended.

The loss function of discriminator and generator are:

$$L(D)=-\mathbb{E}_{x \sim P_r}[D(x)]+\mathbb{E}_{x \sim P_g}[D(x)] \tag{9}$$

$$L(G)=-\mathbb{E}_{x \sim P_g}[D(x)] \tag{10}$$

Eq.9 indicates that the discriminator wants to pull up the scores of the true samples and pull down the scores of the false samples as much as possible, and Eq.10 indicates that the generator wants to pull up the scores of the false samples as much as possible.

2.2 WGAN with gradient penalty

I. Gulrajani et al. improve WGAN with gradient penalty [8], because there are two serious problems with weight clipping.

First, as Eq.9, the discriminator loss wants to maximize the difference between the scores of real and fake samples, however weight clipping independently limits the range of values of each network parameter, in this case the optimal strategy is to make all parameters as extreme as possible, either taking the maximum value or the minimum value, making the discriminator close to a binary network.

Second, weight clipping can lead to an easy accidental gradient vanishing or explosion. The reason is that the discriminator is a multi-layer network, if we set the clipping threshold a little smaller, after each layer of the network, the gradient becomes smaller, multi-layer will be exponentially decaying;

otherwise, if set a little larger, after each layer of the network, the gradient becomes larger, multi-layer will exponentially explode.

We don't really need to impose Lipschitz restrictions on the entire sample space, just focus on the area of the generated sample set, the real sample set, and the area between them. We directly limit the norm of the discriminator gradient to around 1, so the gradient is easy to control and adjust to the right scale.

We start with a random pair of real and fake samples, and a random number of 0-1:

$$x_r \sim P_r, x_g \sim P_g, \epsilon \sim Uniform[0,1] \tag{11}$$

Then randomly interpolate the samples on the line between x_r and x_g.

$$\hat{x} = \epsilon x_r + (1-\epsilon)x_g \tag{12}$$

The final loss function of discriminator is:

$$L(D) = -\mathbb{E}_{x\sim P_r}[D(x)] + \mathbb{E}_{x\sim P_g}[D(x)] + \lambda \mathbb{E}_{x\sim \mathcal{P}_{\hat{x}}}\left[\|\nabla_x D(x)\|_p - 1\right]^2 \tag{13}$$

2.3 WGAIN

M. Friedjungova et al. introduce WGAIN [9] as the Wasserstein modification of GAIN. WGAIN changed the network structure and loss function of discriminators and generators as WGAN did. Although this paper mentions the use of hint matrix and gives parameters, I did not see the corresponding structure in the architecture and pseudo-code, and the code is not open source, so I cannot be sure if it really uses hint mechanism.

2.4 WGAIN with gradient penalty

This is the improvement I proposed based on GAIN.[2] As I discussed in section1.1, I don't think the hint mechanism would work, so I removed it. I keep the structure of generator and discriminator neural networks as in GAIN, each has three layers. I use the hyperbolic tangent activation function ($\tanh()$) instead of $\mathrm{sigmoid}()$ in the output layers of the generator and the discriminator, to make the training model converge faster.

According to the Eqs.11,12 and 13, we can update D using RMSProp method:

$$\nabla_D \frac{1}{k_D}\sum_{j=1}^{k_D}\mathcal{L}_D(D(\tilde{\mathbf{x}}(j)), D(\overline{\mathbf{x}}(j)), \mathbf{m}(j)) + \frac{\lambda}{k_D}\sum_{j=1}^{k_D}\mathcal{L}_{gp}(D(\dot{\mathbf{x}}(j)))$$

$$\nabla_D - \frac{1}{k_D}\sum_{j=1}^{k_D}[\mathbf{m}(j)\odot D(\tilde{\mathbf{x}}(j))] + \frac{1}{k_D}\sum_{j=1}^{k_D}[(\mathbf{1}-\mathbf{m}(j))\odot D(\overline{\mathbf{x}}(j))] + \frac{\lambda}{k_D}\sum_{j=1}^{k_D}\left(\left\|\nabla_{\dot{\mathbf{x}}(j)}D(\dot{\mathbf{x}}(j))\right\|_2 - \mathbf{1}\right)^2$$

where

$$\dot{\mathbf{x}}(j) = \mathbf{m}(j)\odot(\tilde{\epsilon}(j)\odot\tilde{\mathbf{x}}(j)) \oplus ((\mathbf{1}-\mathbf{m}(j))\odot(\mathbf{1}-\tilde{\epsilon}(j))\odot\overline{\mathbf{x}}(j))$$

Then we update G for fixed D:

$$\nabla_G - \frac{1}{k_D}\sum_{j=1}^{k_D}\mathcal{L}_G(C(\overline{\mathbf{x}}(j), \mathbf{m}(j))) + \frac{\alpha}{k_D}\sum_{j=1}^{k_D}\mathcal{L}_M(\tilde{\mathbf{x}}(j), \overline{\mathbf{x}}(j))$$

$$\nabla_G - \frac{1}{k_D}\sum_{j=1}^{k_D}[(\mathbf{1}-\mathbf{m}(j))\odot D(\overline{\mathbf{x}}(j))] + \frac{\alpha}{k_D}\sum_{j=1}^{k_D}\left[\mathbf{m}(j)\odot(\tilde{\mathbf{x}}(j)-\overline{\mathbf{x}}(j))^2\right]$$

The other notations are same as those in the GAIN paper.

[2] `https://github.com/Yihao-Liang/WGAIN_gp`

3 Experiment

I used the same 5 datasets as in GAIN: breast[3], spam[4], letter[5], credit[6] and news[7]. Each dataset was run 12 times on GAIN and WGAIN_gp respectively and RMSE and AUROC were calculated. I obtain and compare the average values after removing the maximum and minimum values.

With the same learning rate, batch size and other parameters, the following are the results of 10,000 epochs of GAIN training and 2,000 epochs of WGAIN_gp training. WGAIN_gp gets close results with a fifth of the GAIN training epochs.

3.1 RMSE

Table 2: Imputation performance in terms of RMSE (Average $\pm$ Std of RMSE)

Algorithm	**Breast**	**Spam**	**Letter**	**Credit**	**News**
WGAIN_gp	**.1252 $\pm$ .0030**	**.0574 $\pm$ .0009**	**.1569 $\pm$ .0003**	**.1737 $\pm$ .0023**	**.2389 $\pm$ .0060**
GAIN	.0731 $\pm$ .0047	.0528 $\pm$ .0008	.1288 $\pm$ .0026	.1630 $\pm$ .0033	.2282 $\pm$ .0118

I did not reproduce the results in the article [1], which may be caused by the parameters. Also my proposed WGAIN_gp did not improve in RMSE.

In addition, I am not sure which news dataset is used in the article, Nabeel gave me different suggestions and the result differs from the article.

3.2 AUROC

After obtaining the imputed data, I randomly select a binary data as the label, use other data for logistic regression, and calculate area Under the ROC(AUROC) curve based on the results obtained from the regression results and the real data.

Table 3: Prediction performance comparison

Algorithm	**Breast**	**Spam**	**Credit**	**News**
WGAIN_gp	**.9913 $\pm$ .0015**	**.9342 $\pm$ .0017**	**.6994 $\pm$ .0026**	**.9911 $\pm$ .0004**
GAIN	.9907 $\pm$ .0022	.9353 $\pm$ .0032	.6971 $\pm$ .0020	.9909 $\pm$ .0008

As Table.3 shows, WGAIN_gp has a slight improvement in AUROC. I did not have more time to try different parameters, but I believe the results will be better if the parameters are set properly. I also did not have much time to test the AUROC with different missing rates.

4 Conclusion

I abadoned the hint mechanism and improved the structure of GAN, especially at the loss function level. Inspired by WGAN, I applied Wasserstein distance to GAIN and obtained similar results. I was not able to come up with a more suitable parameter due to time constraints, but the results are only a little bit lower with the loss with the hint mechanism. WGAN is suitable for high-dimensional samples, especially images (64*64), and can effectively avoid the problem of gradient vanishing due to the non-overlap of P_r and P_g.

Therefore, this work is more applicable to imputation of high-dimensional data than GAIN, especially images and time-series data, which will contribute in medical imaging, EEG and clinical data recording.

Future work includes adjusting the parameters to get better results, verifying the reliability and superiority of WGAIN_gp on high-dimensional data, etc.

[3]`https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29`

[4]`https://archive.ics.uci.edu/ml/datasets/Spambase`

[5]`https://archive.ics.uci.edu/ml/datasets/Letter+Recognition`

[6]`https://archive.ics.uci.edu/ml/datasets/Credit+Approval`

[7]`https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity`

References

[1] Yoon, J., Jordon, J., & Schaar, M. (2018) Gain: Missing data imputation using generative adversarial nets. *In International conference on machine learning*, pp. 5689-5698, PMLR.

[2] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ..., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11), 139-144.

[3] Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., & Paul Smolley, S. (2017). Least squares generative adversarial networks. *In Proceedings of the IEEE international conference on computer vision*, pp. 2794-2802.

[4] Zhang, H., Goodfellow, I., Metaxas, D., & Odena, A. (2019, May). Self-attention generative adversarial networks. *In International conference on machine learning* pp. 7354-7363, PMLR.

[5] Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.

[6] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

[7] Arjovsky, M., Chintala, S., & Bottou, L. (2017) Wasserstein generative adversarial networks. *In International conference on machine learning*, pp. 214-223, PMLR.

[8] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in neural information processing systems, 30*.

[9] Friedjungová, M., Vašata, D., Balatsko, M., & Jiřina, M. (2020). Missing features reconstruction using a wasserstein generative adversarial imputation network. *In International Conference on Computational Science* pp. 225-239. Springer, Cham.

A Proof

A.1 Proof 1

Suppose $M = \frac{1}{2}(P+Q)$, then:

$$JSD(P\|Q) = \frac{1}{2}KL(P\|M) + \frac{1}{2}KL(Q\|M)$$

$$JSD(P\|Q) = \frac{1}{2}\sum p(x)\log\left(\frac{p(x)}{\frac{p(x)+q(x)}{2}}\right) + \frac{1}{2}\sum q(x)\log\left(\frac{q(x)}{\frac{p(x)+q(x)}{2}}\right)$$

$$JSD(P\|Q) = \frac{1}{2}\sum p(x)\log\left(\frac{2p(x)}{p(x)+q(x)}\right) + \frac{1}{2}\sum q(x)\log\left(\frac{2q(x)}{p(x)+q(x)}\right)$$

$$JSD(P\|Q) = \frac{1}{2}\sum p(x)\log\left(\frac{p(x)}{p(x)+q(x)}\right) + \frac{1}{2}\sum q(x)\log\left(\frac{q(x)}{p(x)+q(x)}\right) + \log 2$$

this is because $\sum p(x) = \sum q(x) = 1$.

As Fig.A.1 shows, let $p(x)$ be the probability when P_r contains x, $q(x)$ be the probability when P_g contains x. There is almost no overlap between the two distributions.

when $x \geq 5$, $p(x) \to 0$:

$$JSD(P\|Q) = \frac{1}{2}\sum 0\log\left(\frac{0}{0+q(x)}\right) + \frac{1}{2}\sum q(x)\log\left(\frac{q(x)}{0+q(x)}\right) + \log 2 = 0 + \log 2 = \log 2$$

The same as $x < 5$, $q(x) \to 0$. Therefore:

$$\forall x \in R, JSD(P\|Q) = \log 2$$

□

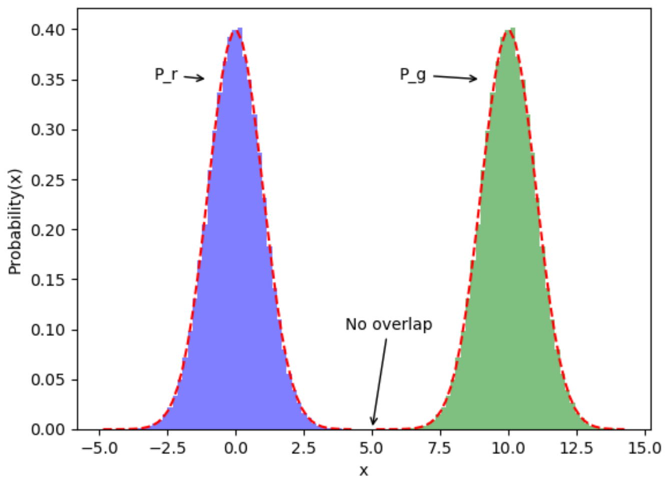

Figure 1: Two normal distributions without overlap.

A.2 Proof 2

According to Eqs.3,4and6, We can transform the KL divergence into a form of D^*.

$$KL\left(P_g \| P_r\right) = \mathbb{E}_{x\sim P_g}\left[\log \frac{P_g(x)}{P_r(x)}\right]$$

$$KL\left(P_g \| P_r\right) = \mathbb{E}_{x\sim P_g}\left[\log \frac{P_g(x)/\left(P_r(x)+P_g(x)\right)}{P_r(x)/\left(P_r(x)+P_g(x)\right)}\right]$$

$$KL\left(P_g \| P_r\right) = \mathbb{E}_{x\sim P_g}\left[\log \frac{1-D^*(x)}{D^*(x)}\right]$$

$$KL\left(P_g \| P_r\right) = \mathbb{E}_{x\sim P_g} \log\left[1-D^*(x)\right] - \mathbb{E}_{x\sim P_g} \log D^*(x)$$

Then

$$\mathbb{E}_{x\sim P_g}\left[-\log D^*(x)\right] = KL\left(P_g \| P_r\right) - \mathbb{E}_{x\sim P_g} \log\left[1-D^*(x)\right]$$

$$\mathbb{E}_{x\sim P_g}\left[-\log D^*(x)\right] = KL\left(P_g \| P_r\right) - 2JS\left(P_r \| P_g\right) + 2\log 2 + \mathbb{E}_{x\sim P_r}\left[\log D^*(x)\right]$$

Note that the last two terms of the above equation do not depend on the generator G, therefore minimizing Eq.3 is equivalent to minimizing:

$$KL\left(P_g \| P_r\right) - 2JS\left(P_r \| P_g\right)$$

□

B Parameters

B.1 GAIN

miss rate = 0.2

batch size = 128

hint rate = 0.9

α = 100

iterations = 10000

B.2 WGAIN_gp

n critic = 5 (number of additional iterations to train the critic)

batch size = 128

miss rate = 0.2

λ_{gp} = 10

iterations = 2000

α = 100

learning rate = 1×10^{-3}

decay = 0.9

momentum = 0.000

ϵ = 1×10^{-8}

B.3 Logistic Regression

test size:train size = 3:7

iteration max = 30000