Final Report

Improving Credit Card Fraud Detection using Generative Adversarial Networks

Group 4 Team Member: Hao Ning, Jun Ying

Problem Statement

As credit card transactions have become the mainstream consumption pattern, the number of credit card frauds has increased dramatically. Even if users are finally compensated, they often spend a lot of time and even money in the process. Therefore, credit card fraud detection systems are essential for banks and financial institutions to minimize their losses nowadays.

In this study, we will first use traditional machine learning algorithms for classification training. However, relative to the number of transactions, the proportion of credit card fraud is very small. Traditional machine learning tends to be biased towards the mainstream category (non-fraud transaction) in the imbalanced data set of classification. Therefore, we will use Generative Adversarial Networks (GAN) to generate artificial credit card fraud data to oversample the data set. Since the data is unbalanced, accuracy is not a fair evaluation of the model performance, thus statistical metrics such as precision, recall, f1 score, ROC AUC score will be compared for different models.

GAN is a generative model in which two neural networks (generator and discriminator) compete and improve each other. The generator keeps generating more and more "real" fake data to deceive the discriminator. However, the discriminator needs to distinguish between the data generated from the generator and the original data. Unlike using GAN working with images and evaluating the generated images by eyes, we are working with tabular data. We use boxplot of the data distribution to evaluate the generated data to see the data range and how well they overlap with the original data.

Through the artificial credit card fraud data generated by GAN, we will well oversample the data set to improve the accuracy of the classification model. In this project, we implement different GAN models for detailed performance comparison, including vanilla GAN[1], Wasserstein GAN(WGAN)[2], WGAN_GP[3] Balancing GAN (BAGAN)[4] and Boundary Equilibrium GAN (BEGAN)[5].

Data Description

The data set contains 284,807 transactions made by European cardholders within two days through credit cards in September 2013. However, there are only 492 frauds in all transactions, and the class of fraud accounts for 0.172% of all transactions.

Due to confidentiality issues, features V1, V2, … V28 are the principal components obtained by PCA transformation; ‘Time’ is sorted in time series in seconds; ‘Amount’ is the transaction Amount; and 'Class' is the response variable and it takes value 1 in case of fraud and 0 non-fraud.

---

[1] Goodfellow, Ian, etc. (2014). Generative Adversarial Networks
[2] Wasserstein GAN [1701.07875] Wasserstein GAN
[3] Improved Training of Wasserstein GANs https://arxiv.org/pdf/1704.00028
[4] BAGAN [1803.09655] BAGAN: Data Augmentation with Balancing GAN
[5] BEGAN: Boundary Equilibrium Generative Adversarial Networks [1703.10717] BEGAN: Boundary Equilibrium Generative Adversarial Networks

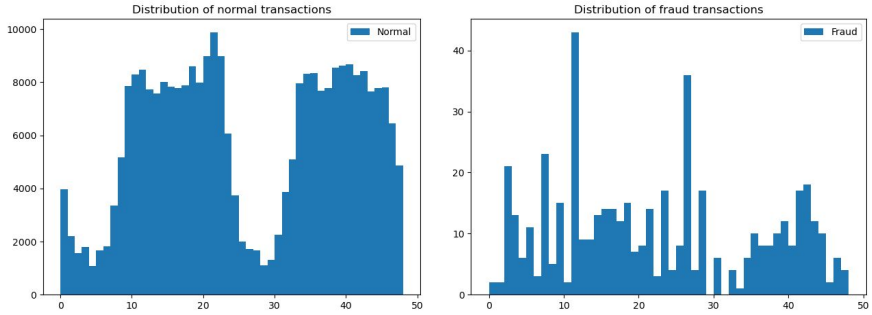
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0

Link to the data: [Credit Card Fraud Detection]

EDA

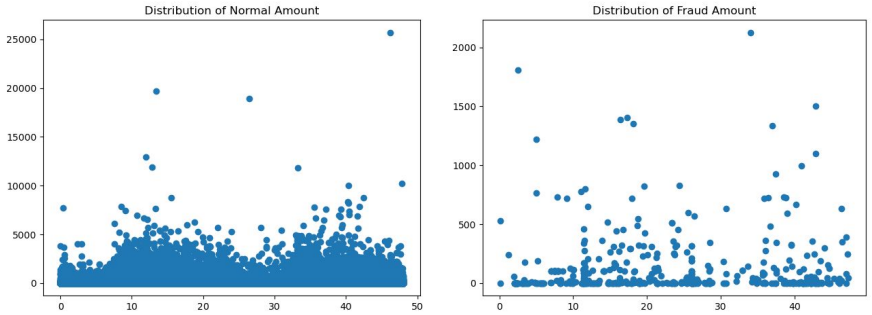
Among the 30 features, only time and amount are data not processed by PCA. Therefore, we start with these two features.

Time



The time span of this data set is two days, so we can see from the chart that normal transactions have seasonality. However, fraud transactions are irregular, only peaking at 12 noon (12h) on the first day and 2 am (26h) on the second day.

Amount



We can find from this scatter plot that the number of super-large transactions is very small. In comparison, the largest amount of normal transactions is over €25,000. However, the largest amount of fraud transactions is only €2,000.

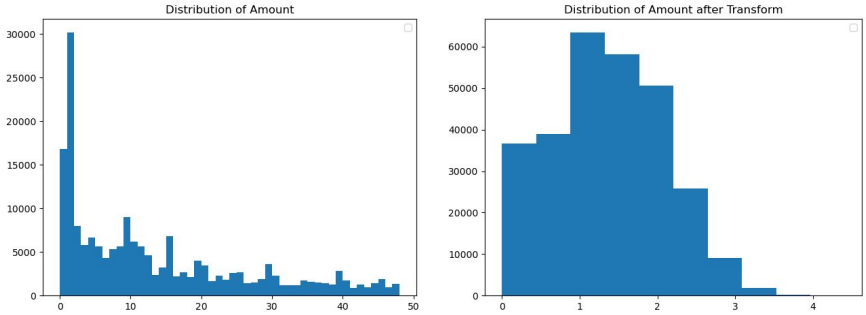
Data Preprocessing

'Time' is sorted in time series in seconds change to hours

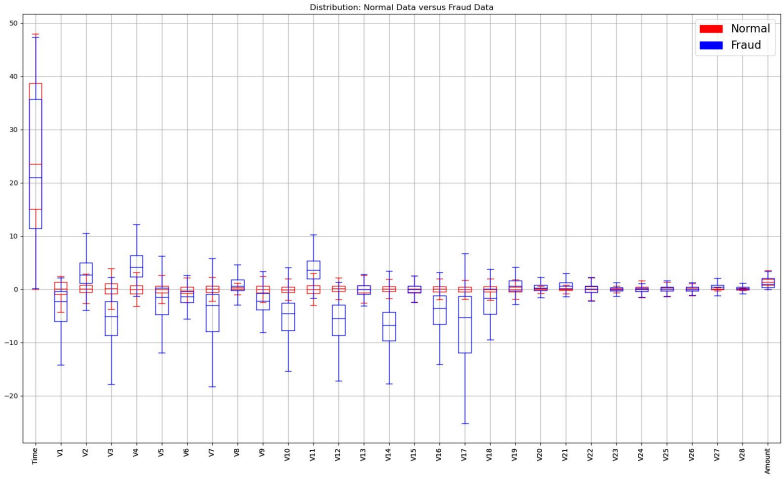
- df['Time'] = (df['Time'].values / 3600)

'Amount' is larger than other features, so we have to reduce its scope.

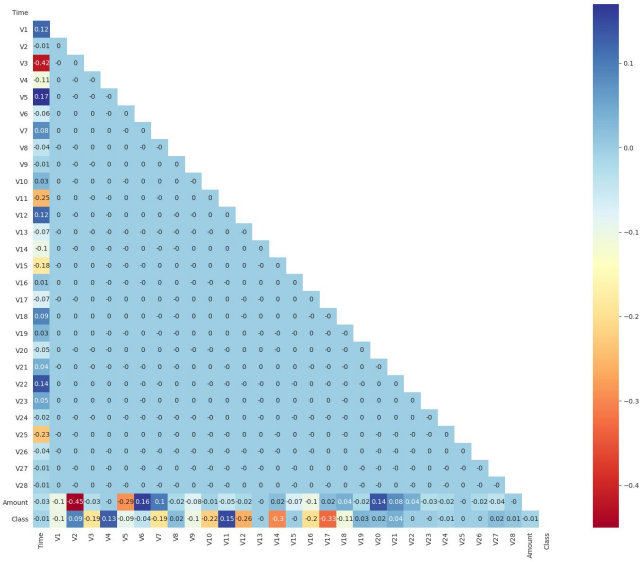
- df['Amount'] = np.log10(df['Amount'].values + 1)



Data Distribution & Correlation



We can see that even in features V1 to V28, their distribution is very different.

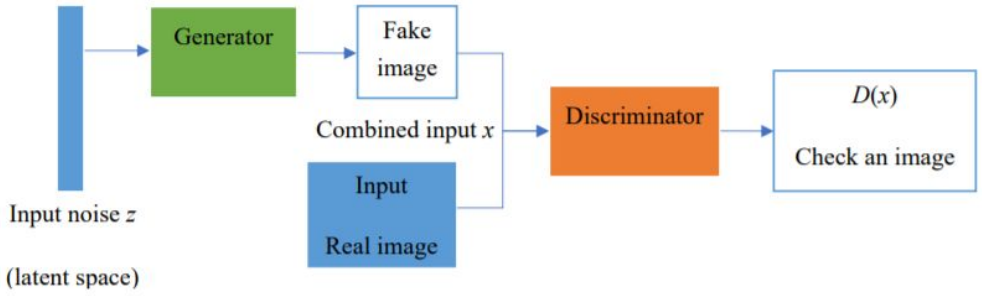


There is not much correlation between features, only from -0.45 to 0.17.

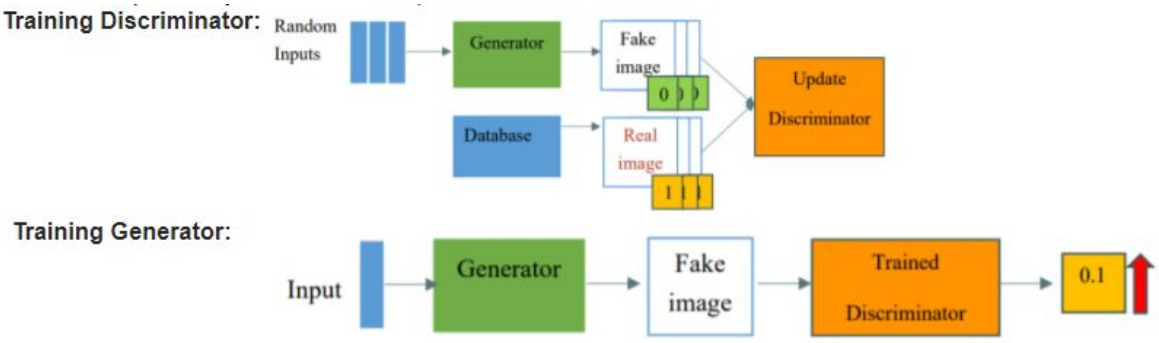
GAN Theory

GAN is composed of the generator and the discriminator, which are against each other in a zero-sum game. Typically, it is used to generate virtual images. For example, a photo-trained GAN can generate new photos, which can fool humans who will think these new photos are real.

$$min_D max_G V(G,D) = \mathbb{E}_x \log\big(D(x)\big) + \mathbb{E}_z \log\Big(D\big(G(z)\big)\Big)$$



The generator generates images while the discriminator evaluates them. The generator improves its performance to confuse the evaluation of discriminator, while the discriminator improves its performance to distinguish generated images and real images. This is a minimax problem, which means if the generator improved, the performance of the discriminator would definitely get worse.



The core idea of GAN is based on the 'indirect' training through the dynamically updated discriminator.[6] Fundamentally, this means that the generator is trained to fool the discriminator.

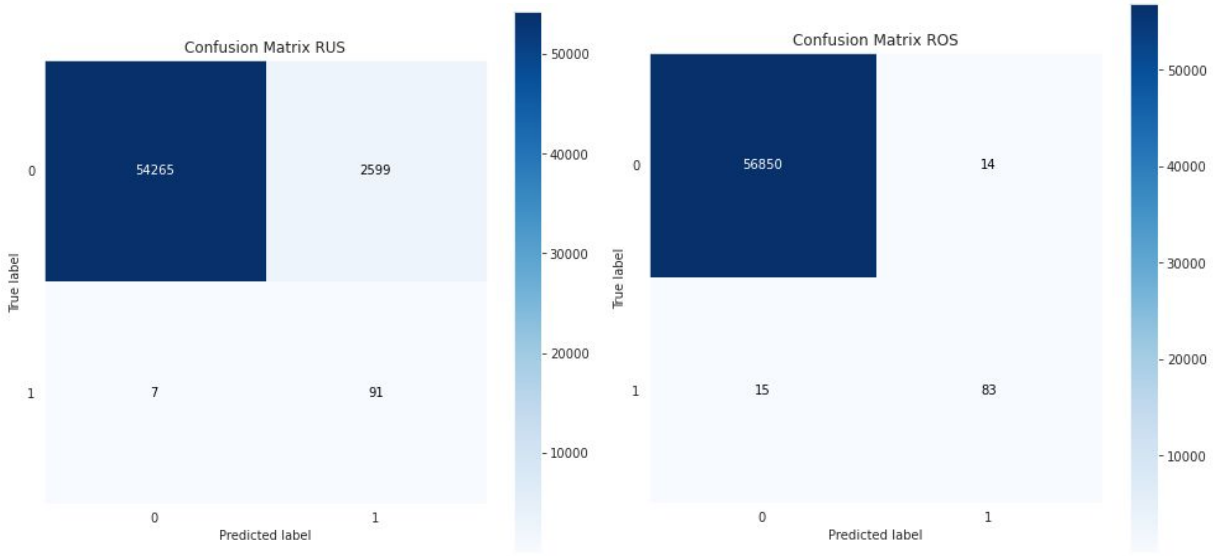
Base Model

Base Model is developed by basic machine learning approach:

[6] Vanilla GAN (GANs in computer vision: Introduction to generative learning): AI Summer

1. Split the data with train set 80% 227845 (394 fraud) and test set 20% 56962 (98 fraud), and labels are stratified
2. Perform Random Under Sampling (RUS) & Random Over Sampling (ROS) on train set
3. Use GridsearchCV to find the best parameters for XGBoostClassifier
4. Predict with best parameters
5. Test performance evaluation, results are shown below

	RUS 1 394 0 394	ROS 1 227451 0 227451
Accuracy	0.954250	0.999491
Precision	0.033829	0.855670
Recall	0.928571	0.846939
F1 score	0.065280	0.851282
ROC AUC score	0.941433	0.923346



Confusion Matrix of RUS left & ROS right

By comparing the model performance, RUS has very high false positives, therefore obviously we will use ROS as our base model.

GAN

Vanilla gan deals with images with data normalized to (0, 1), but we are dealing with tabular data, so tanh is removed. Also, batch normalization is removed since it yields bad results.

The noise has the size of 32 and our feature length is 30. The GAN generator & discriminator structure is shown below:

Generator

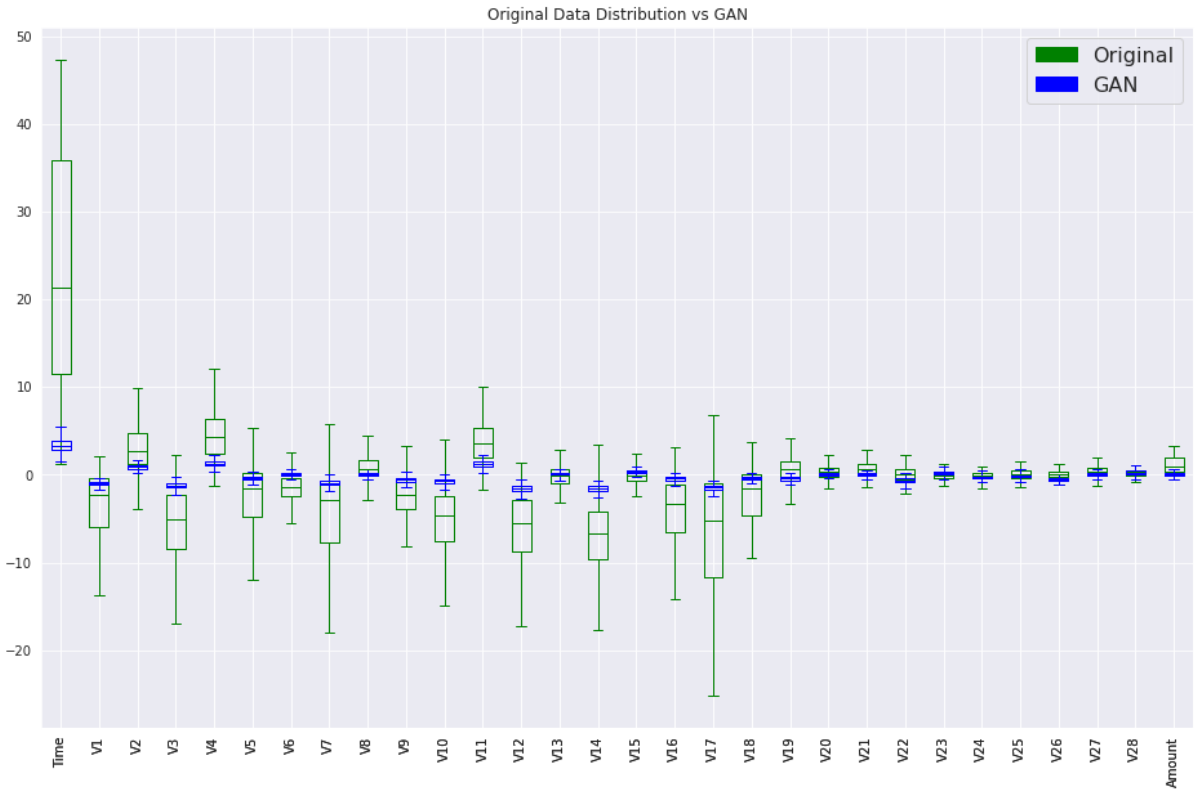
Input	Output size
(32, 1) **Noise**	(64,1)
(64, 1)	(128, 1)
(128, 1)	(256, 1)
(256, 1)	(30, 1) **Features**

Discriminator

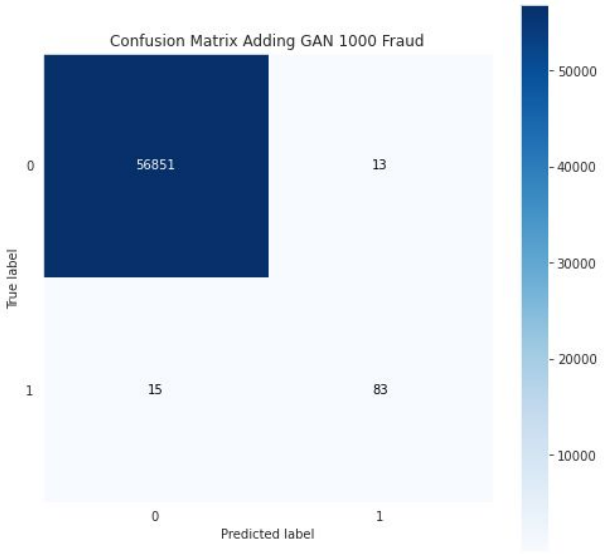
Input	Output size
(30, 1) **Features**	(256,1)
(256, 1)	(128, 1)
(128, 1)	(64, 1)
(64, 1)	(1, 1) **real data or generated data label**

GAN Performance

1000 of GAN generated data is added to the train set, the boxplot comparison of the data distribution shows that, GAN is only generating very low spectrum data, which is a common issue in GAN called mode collapse. Also, the generated data showed poor overlap with the original data, for example, the mean of the generated data are kind of deviated from the mean of original data.



Data distribution of GAN generated data & original fraud data from train set



	Base	GAN
Accuracy	0.999491	0.999491
Precision	0.855670	**0.864583**
Recall	0.846939	0.846939
F1 score	0.851282	**0.855670**
ROC AUC score	0.923346	0.923355

Confusion Matrix of GAN adding 1000 fraud & Statistical Score

There's a slight improvement on model performance observed with GAN generated data. This is due to mode collapse problems (generating low spectrum data) as we showed before in the data distribution. We will implement other improved GAN algorithms next to see if there's any improvement for fraud detection.

WGAN

In GAN, the loss measures how well the generator fools the discriminator (since the output is a probability/label) rather than the measurement of quality. Thus, vanilla gan usually suffer problems like mode collapse (generating low spectrum data shown before) and gradient vanishing. Gradient vanishing can be explained by math shown below.

JS Convergence[7]

Loss function of GAN is equivalent to JS convergence (when D at optimality). The problems in GAN can be explained with math.

First, what is the best value for D:

$$L(G,D) = \int_x \Big(p_r(x)\log(D(x)) + p_g(x)\log(1-D(x)) \Big) dx$$

Let

$$\tilde{x} = D(x), A = p_r(x), B = p_g(x)$$

$$\begin{aligned} f(\tilde{x}) &= Alog\tilde{x} + Blog(1-\tilde{x}) \\ \frac{df(\tilde{x})}{d\tilde{x}} &= A\frac{1}{ln10}\frac{1}{\tilde{x}} - B\frac{1}{ln10}\frac{1}{1-\tilde{x}} \\ &= \frac{1}{ln10}(\frac{A}{\tilde{x}} - \frac{B}{1-\tilde{x}}) \\ &= \frac{1}{ln10}\frac{A-(A+B)\tilde{x}}{\tilde{x}(1-\tilde{x})} \end{aligned}$$

The best value of the discriminator is calculated by set the derivative to 0 :

[7] Hongyi Lee, GAN 2018
https://www.youtube.com/watch?v=DQNNMiAP5lw&list=PLJV_el3uVTsMq6JEFPW35BCiOQTsoqwNw
From GAN to WGAN https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

$$D^*(x) = \tilde{x}^* = \frac{A}{A+B} = \frac{p_r(x)}{p_r(x)+p_g(x)} \in [0,1].$$

Once G is trained to optimality, Pg = Pr, therefore D = 0.5!

In other words, assuming that both D and G are optimized, discriminator can only flip a coin to tell the performance of the generator, although it's already generating samples as good as real ones.

In reality, Pg and Pr are always not overlapping for the following reasons:

1. Both Pg and Pr are low-dimensional manifold in high-dimensional space
2. Sampling

Now, the problem of JS divergence can be explained mathematically.

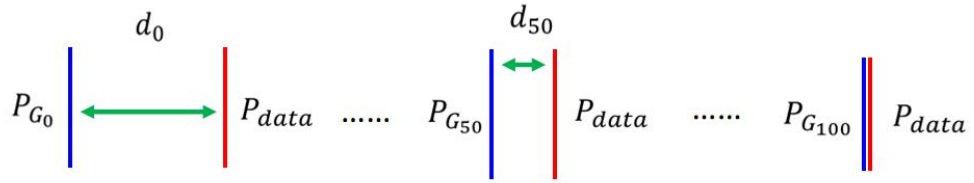
When 2 distribution are not overlapping:

$$KL(P_1||P_2) = \mathbb{E}_{x\sim P_1} \log \frac{P_1}{P_2} \qquad JS(P_1||P_2) = \frac{1}{2}KL(P_1||\frac{P_1+P_2}{2}) + \frac{1}{2}KL(P_2||\frac{P_1+P_2}{2})$$

$$D_{JS}(\mathrm{P_r}, \mathrm{P_g}) = 1/2(\sum_{\mathrm{x=0,\,y\sim U(0,1)}} 1 \cdot \log\frac{1}{1/2} + \sum_{\mathrm{x=0,\,y\sim U(0,1)}} 1 \cdot \log\frac{1}{1/2}) = \log 2$$

The JS convergence is always log2, so there will be no gradient for optimization.

Comparing Wasserstein distance to JS convergence:



$JS(P_{G_0}, P_{data}) = log2$ $\quad$ $JS(P_{G_{50}}, P_{data}) = log2$ $\quad$ $JS(P_{G_{100}}, P_{data}) = 0$

$W(P_{G_0}, P_{data}) = d_0$ $\quad$ $W(P_{G_{50}}, P_{data}) = d_{50}$ $\quad$ $W(P_{G_{100}}, P_{data}) = 0$

Wasserstein distance can effectively calculate the difference between Pg and Pr even when they are not overlapping. To get Wasserstein distance, the objective function will be

$$V(G,D) = \max_{D \in 1-Lipschitz} \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)]\}$$

1-Lipschitz function means:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|$$

For 1-Lipschitz function, it should be everywhere continuously differentiable. The output change should be less or equal to the input change.

Since D has to be a smooth function, WGAN proposed that using weight clipping (force the parameters w between c and -c) to provide a constraint, otherwise, as shown bottom right, the output of real data will become +infinity, the generated data will be going -infinity, then D will not converge. Note that D with weight clipping here is not a strict 1-Lipschitz function, it's an 'engineering' approach (also for WGAN-GP).

WGAN Algorithm[8] and Implementation

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Two significant improvement by introducing WGAN:

- No sign of mode collapse in experiments, wider range of data
- The generator can still learn when the critic performs well

[8] https://arxiv.org/pdf/1701.07875.pdf

The implementation of WGAN needs a few adjustments on vanilla GAN. There's no sigmoid function at output of the discriminator, thus the output is a scalar score, that can be interpreted as how real the input images are (in our case, the tabular input features), rather than a probability. Regarding the generator G, we want to maximize the D's output for its fake instances. The changes are summarized as follows:

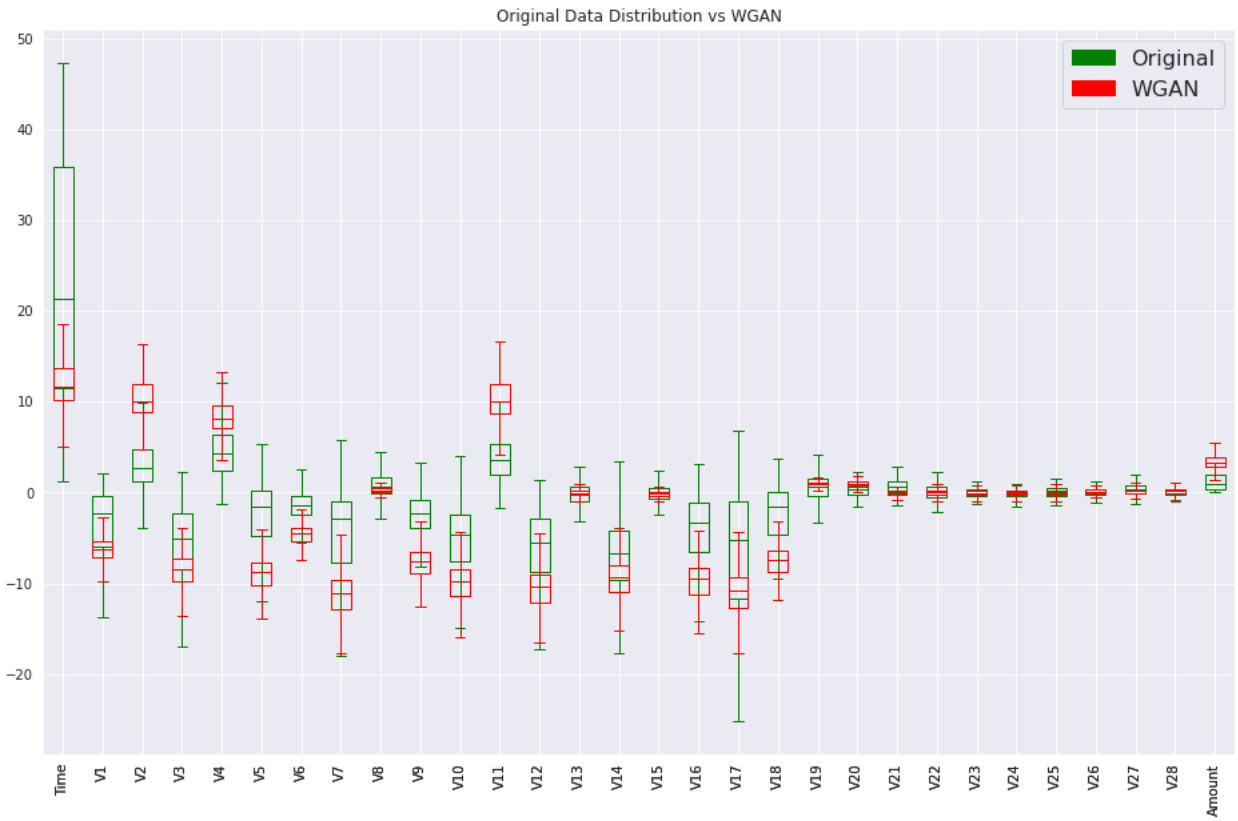
- Loss function, no log
- D: no sigmoid, train D more than G
- Clip the weight of D (-c,c), if w>c, w=c, if w<-c, w=-c
- Use RMSProp

Hyperparameters

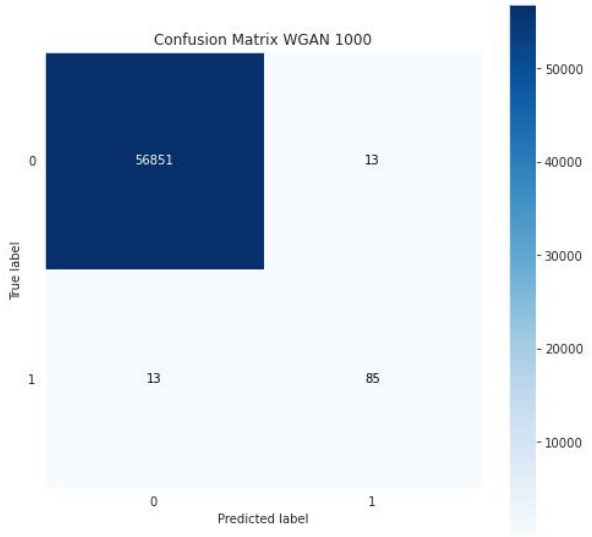
- Clip Weight at (-0.01,0.01), Batch size 128, Learning rate 0.00001

WGAN Performance

By adding 1000 WGAN generated fraud to train set, data distribution and the statistical analysis of the model performance are shown below:



Data distribution of WGAN generated data & original fraud data from train set



	WGAN
Accuracy	**0.999544**
Precision	**0.867347**
Recall	**0.867347**
F1 score	**0.867347**
ROC AUC score	**0.933559**

Confusion Matrix of WGAN adding 1000 fraud & Statistical Score

The performance of the fraud detection is improved with the help of WGAN, this makes sense since the generated fraud data showed a wider range and improved overlap with the original fraud data distribution compared to GAN.

WGAN_GP

WGAN_GP development:

WGAN-GP is an improved version of WGAN, the difference between WGAN-GP and WGAN is that WGAN-GP uses gradient penalty instead of weight clipping. WGAN weight clipping does not strictly fullil the 1-Lipschitz function requirement as shown below, when K =1 is a 1-Lipschitz function, it has strong uniform continuity.

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|.$$

A differentiable function is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere**,** therefore WGAN_GP constrain the weight more effectively.

WGAN_GP algorithm and implementation[9]

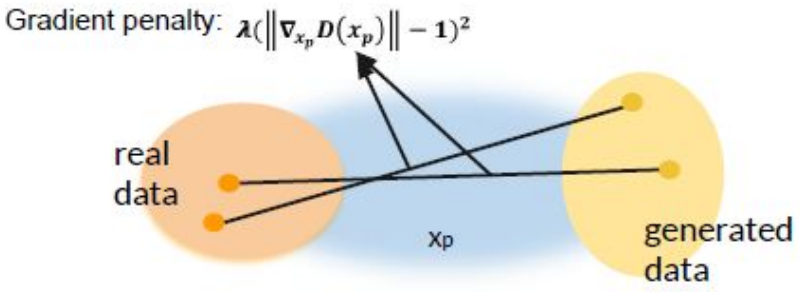
Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Changes compared to WGAN, WGAN_GP uses gradient penalty & Adam optimizer

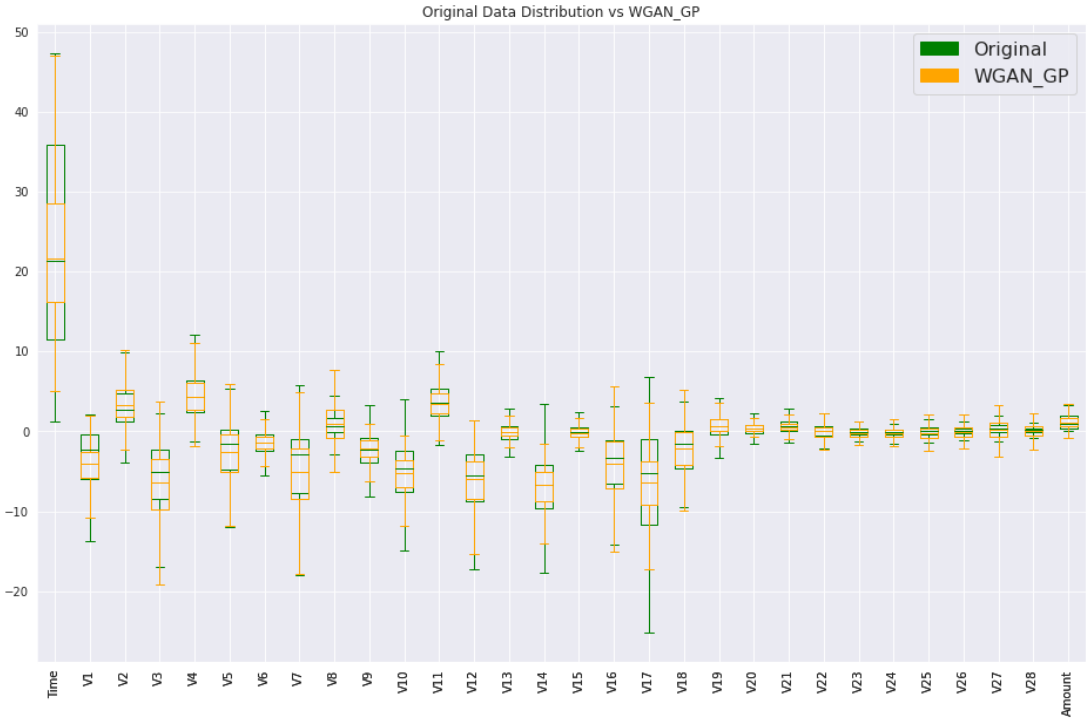
It's hard to get gradients with norm at most 1 everywhere, so we interpolate between real and generated samples. The points interpolated between real and generated data should have a gradient norm of 1. WGAN-GP will apply a penalty if the gradient norm moves away from its target norm value 1.



Training Parameters: ADAM(lr=0.00001), batch_size=128, gp_lambda=5

After generating 1000 'Fraud', we visually compare the data distribution using boxplot as shown below.

[9] Improved Training of Wasserstein GANs https://arxiv.org/pdf/1704.00028



	WGAN_GP
Accuracy	0.999596
Precision	0.894737
Recall	0.867347
F1 score	0.880829
ROC AUC score	0.933586

Confusion Matrix of WGAN_GP adding 1000 fraud & Statistical Score

Further improvement on model performance is observed by using WGAN_GP, since the generated fraud data showed an even better overlap with the original fraud data distribution, WGAN_GP is generating more reliable ‘fraud’ data.

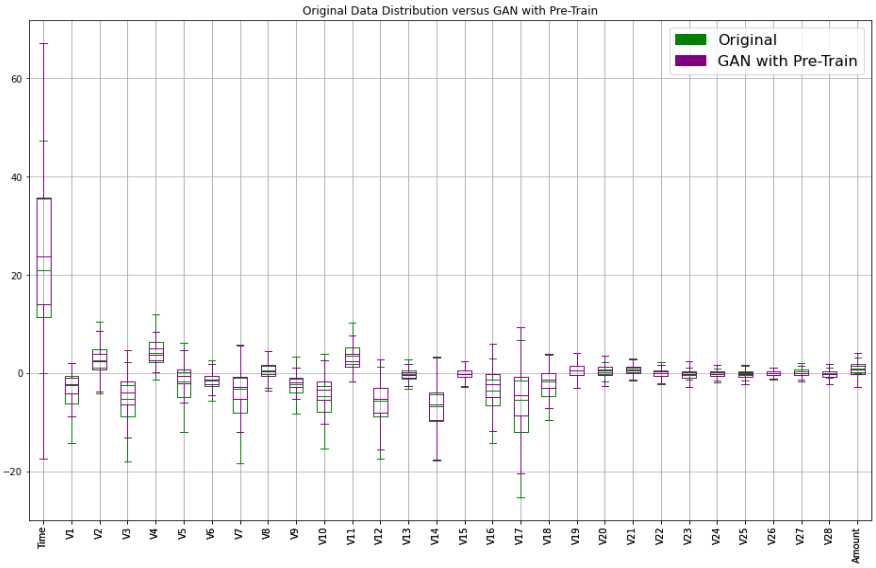
Autoencoder

An autoencoder is an artificial neural network that is used to learn effective data encoding in an unsupervised manner. The purpose of an autoencoder is to learn the representation (encoding) of a set of data by training the network to ignore the signal "noise".

GAN is actually a variant of autoencoder, so we decided to use autoencoder to pre-train GAN. Therefore, we initialized GAN using the parameters of the trained autoencoder.

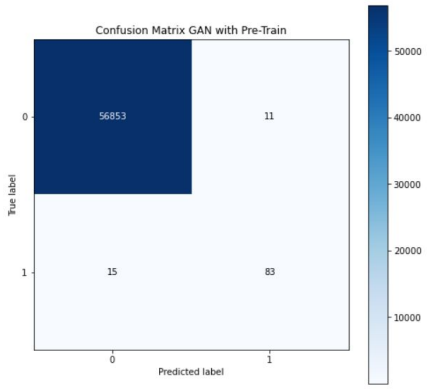
GAN with Pre-Train Performance

We generated 1000 fraud data, and compared with the distribution of original fraud data.



We can see from the graph that the distribution of the generated data covers the distribution of original data well. However, there are some negative values of time and amount.

	GAN	GAN + AE
Accuracy	**0.999508**	**0.999544**
Precision	**0.864583**	**0.882979**
Recall	**0.846939**	**0.846939**
F1 score	**0.855670**	**0.864583**
ROC AUC score	**0.923355**	**0.923373**



After pre-training, it significantly improves the performance of GAN.

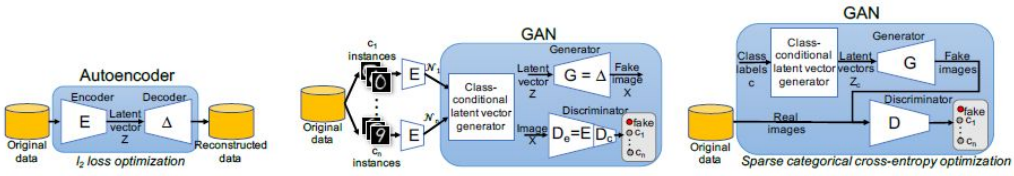
BAGAN

BAGAN is an augmentation tool to restore the dataset balance by generating new minority-class data. It can learn the underlying features of the specific classification problem starting from all data and then to apply these features for the generation of new minority-class data.

BAGAN also uses an autoencoder to initialize the model so that it can far from mode collapse.

GAN initialization

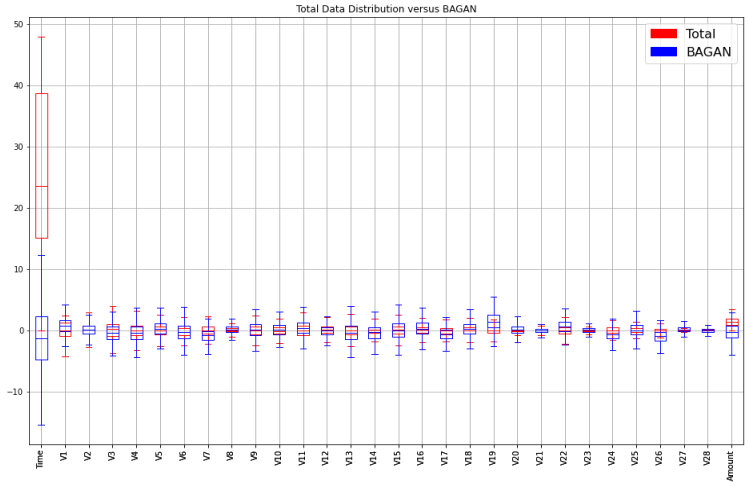
Differently from the other GAN, the generator and the discriminator have explicit class knowledge. During the training, generator is asked to generate data for different classes, and discriminator is asked to label the data either as fake or class label. Its combined class label and latent vector, and then input to the generator to generate specific classes.



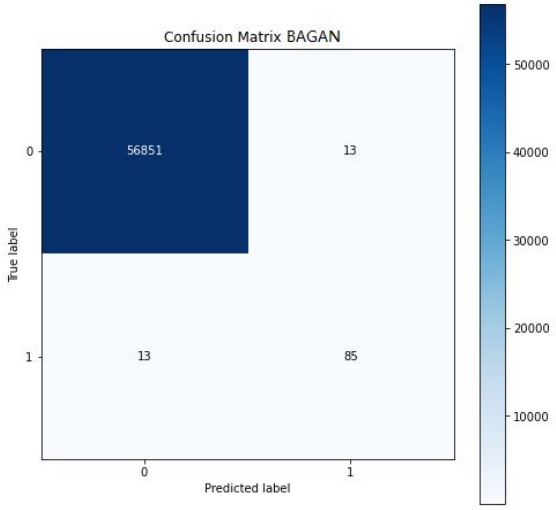
(a) Autoencoder training. (b) GAN initialization. (c) GAN training.

BAGAN Performance

We generated 1000 fraud data, and compared with the distribution of total original data.



Except for time and amount, the generated fraud data are the same as the total data. Because the number of fraud data is too small, we think it learns too much from the normal transactions. In addition, It might also because there is no similarity between fraud and normal transactions, the generator cannot learn useful information from normal data.



	GAN	BAGAN
Accuracy	0.999508	0.999544
Precision	0.864583	0.867347
Recall	0.846939	0.867347
F1 score	0.855670	0.867347
ROC AUC score	0.923355	0.933559

BAGAN performs better than GAN but worse than WGAN_GP.

BEGAN

BEGAN is an equilibrium concept that balances the power of the discriminator against the generator during training.[10] It combines the loss derived from the Wasserstein distance to train the autoencoder-based model (by using autoencoder as the discriminator.).

BEGAN Algorithm

The loss function for training autoencoder is:

$$\mathcal{L}(v) = |v - D(v)|^{\eta} \text{ where } \begin{cases} D : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_x} & \text{is the autoencoder function.} \\ \eta \in \{1, 2\} & \text{is the target norm.} \\ v \in \mathbb{R}^{N_x} & \text{is a sample of dimension } N_x. \end{cases}$$

Discriminator loss & Generator loss:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x; \theta_D) - \mathcal{L}(G(z_D; \theta_G); \theta_D) & \text{for } \theta_D \\ \mathcal{L}_G = -\mathcal{L}_D & \text{for } \theta_G \end{cases}$$

[10] BEGAN: Boundary Equilibrium Generative Adversarial Networks [1703.10717] BEGAN: Boundary Equilibrium Generative Adversarial Networks

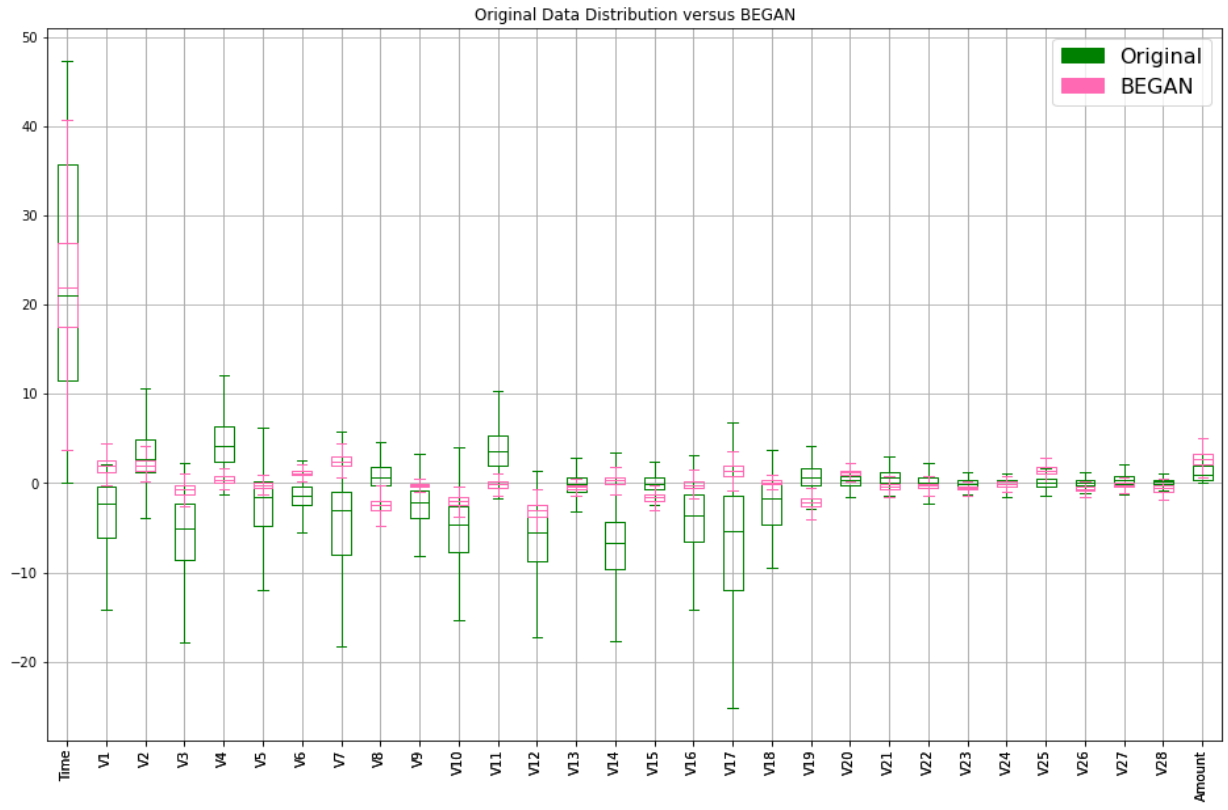
Where θ_D and θ_G is the parameters of Discriminator and Generator, each updated by minimizing the losses L_D and L_G; z_D and z_G are from $z \in [-1, 1]^{N_z}$, which uniform random samples of dimension N_Z.
For training model, the objective of BEGAN is:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t.\mathcal{L}(G(z_D)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z_G)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k(\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))) & \text{for each training step } t \end{cases}$$

BEGAN use Proportional Control Theory to control the equilibrium. $k_t \in [0, 1]$ is used to control the importance of $L(G(z_D))$ during gradient descent. During the training process, it uses λ_k (the learning rate of k_t) to dynamically update k_t.

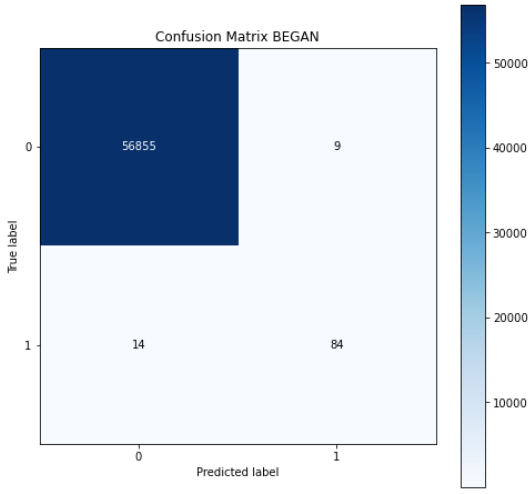
BEGAN Performance

We generated 1000 fraud data, and compared with the distribution of original fraud data.



The generated data covered about half of the fraud data spectrum.

	BAGAN	BEGAN
Accuracy	**0.999544**	**0.999596**
Precision	**0.867347**	**0.903226**
Recall	**0.867347**	**0.857143**
F1 score	**0.867347**	**0.879581**
ROC AUC score	**0.933559**	**0.928492**



The performance of BEGAN is excellent, only slightly worse than WGAN-GP.

Conclusions

	Base	GAN	WGAN	**WGAN_GP**	GAN + AE	BAGAN	**BEGAN**
Accuracy	0.999491	0.999491	0.999544	**0.999596**	0.999544	0.999544	**0.999596**
Precision	0.855670	0.864583	0.867347	**0.894737**	0.882979	0.867347	**0.903226**
Recall	0.846939	0.846939	0.867347	**0.867347**	0.846939	0.867347	**0.857143**
F1 score	0.851282	0.855670	0.867347	**0.880829**	0.864583	0.867347	**0.879581**
ROC AUC score	0.923346	0.923355	0.933559	**0.933586**	0.923373	0.933559	**0.928492**

Comparison of all GAN models performance

By comparing the statistical score of all the different models, WGAN_GP and BEGAN have the highest F1 score, on the other hand, WGAN_GP has the highest recall and ROC AUC score, indicating WGAN_GP has the best overall model performance with different threshold, BEGAN has highest precision but low recall.

For real world problems, numbers are not the only thing to look at. In fraud detection(similarly in sick patient detection, risk detection etc), the cost of false negative (FN) is usually higher than false positive (FP)[11]. We don't want to label/predict a fraudulent transaction (TP) as non-fraudulent (FN). Because for FP, the mistake in classification will be identified in further investigations. To control the fraud detection cost, the system should take into account both the cost of fraudulent behavior that is detected and the cost of preventing it.

In this project, we demonstrated that GANs work well with tabular data with proper modification. Vanilla GAN slightly improved fraud detection since it only generated low spectrum data due to mode collapse, while for other improved GAN models, they are able to generate a wider range of data and overlap well with the original fraud data. WGAN_GP and BEGAN perform best among all GANs. We also showed that using GAN as an oversampling strategy has great potential in credit card fraud detection and extremely imbalanced dataset.

[11] A Survey of Credit Card Fraud Detection Techniques: Data and Technique Oriented Perspective

Individual Work

Hao Ning: Base model, GAN, WGAN & WGAN_GP

Jun Ying: EDA, GAN+AE, BAGAN & BEGAN

References

1. Goodfellow, Ian, etc. (2014). Generative Adversarial Networks
2. Wasserstein GAN [1701.07875] Wasserstein GAN
3. Improved Training of Wasserstein GANs https://arxiv.org/pdf/1704.00028
4. BAGAN [1803.09655] BAGAN: Data Augmentation with Balancing GAN
5. BEGAN: Boundary Equilibrium Generative Adversarial Networks [1703.10717] BEGAN: Boundary Equilibrium Generative Adversarial Networks
6. Hongyi Lee, GAN 2018 https://www.youtube.com/watch?v=DQNNMiAP5lw&list=PLJV_el3uVTsMq6JEFPW35BCiOQTsoqwNw
7. From GAN to WGAN https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html
8. Keras GAN https://github.com/eriklindernoren/Keras-GAN
9. A Survey of Credit Card Fraud Detection Techniques: Data and Technique Oriented Perspective