Synthetic Time Series Data Creation using Generative Adversarial Networks (GAN)

Brigham Freeman

March 2025

1 Introduction

As artificial intelligence continues to expand its footprint in the business world, there are a number of benefits that businesses see from implementing artificial intelligence solutions. Increased efficiency, optimized operations, cost savings, and competitive advantages are all results of AI implementations. However, AI models often require large datasets for effective training. For many businesses, datasets are not readily available, or don't contain the quality information that is necessary for deep learning models. In a survey by Deloitte, 77% of businesses have concerns over the quality of their data, with 91% of respondents saying the quality of available data negatively impacts business decisions [1].

To ensure that there is enough data to train models, many data scientists and machine learning engineers turn to creating synthetic data for training. This has several advantages, including reduced time to annotate and process data, simulation for edge cases, and higher data privacy. While synthetic data can be useful for training models, it does not always capture the complexity real-world data contains. There are also quality and accuracy concerns, as the algorithms that create synthetic data may not always be perfectly accurate.

Generative Adversarial Networks (GANs for short) are a framework for neural networks specifically tailored for synthetic data creation. GANs can be used to create high-quality, realistic datasets when real data isn't available. They can also be used to "replicate" existing high quality data, increasing the amount of existing samples. They work by training two neural networks, a *generator* and a *discriminator* to compete with each other. The generator will generate data that is based off of real data samples, while the discriminator learns to classify data from the generator as real or fake [2]. This adversarial nature, which is where GANs gained their name, helps improve both networks. Over time, this leads to a network that creates data that is so realistic, the discriminator cannot tell the difference between real and fake data. This creates a dataset that is a close approximation for real data. While computationally expensive to create, this new synthetic data can be used to train other AI models on pieces of information that can be indistinguishable from real data. This empowers businesses to "replicate" their high quality data, addressing the concerns that low quality data presents.

GAN models have been widely applied to many different sectors, including medical, finance, and technology. In the medical field in particular, GAN models are an extremely powerful tool for creating data sets. Since there are strict controls around data privacy, synthetic data that replicates real data allows for AI systems to learn

effectively while protecting sensitive patient data [3]. In the tech sector, GAN models allow for high-quality data that is able to more effectively predict cybersecurity incidents than purely synthetic data [4].

2 Theoretical Background

While the mathematics that govern most neural networks are very complicated, GAN models are even more complex. GAN models utilize game theory, optimization, and probability theory to learn. These networks are a *two-player zero-sum game*. In these types of games, one player's loss is another's gain, meaning the net result is 0. The main objective of this game is for the generator, G, and discriminator, D, to improve over time while competing with each other. The generator G maps inputs $p_{\text{model}}(x)$ to the data space $G(x;\theta_g)$. G is represented by a multilayer neural network with parameters θ_g. The discriminator D is another multilayer neural network $D(x,\theta_d)$ that outputs a single value. The value output is the probability of x being real data. In essence, D learns the probability distributions between the fake and real data sets.

Let $p_{\text{data}}(x)$ be the distribution of real data, and $p_{\text{model}}(x)$ be the distribution of data produced by the generator. The generator and discriminant use the min-max game given by

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{x\sim p_{\text{model}}}[\log(1-D(G(x)))]$$

The objective function $V(G,D)$ is made up of two components. The first is $\mathbb{E}_{x\sim p_{data}}[\log D(x)]$ [1] [2]. This takes a randomly sampled x from the distribution of real data, p_{data}. The discriminator $D(x)$ returns the probability of the sampled value x being real. The discriminator tries to maximize this value by outputting values close to 1 for real data.

The second component, $\mathbb{E}_{x\sim p_{model}}[\log(1-D(G(x)))]$, takes an x sampled from the probability distribution of the fake data p_{model}. The generator, $G(x)$, tries to create a fake data sample from a random input. The discriminator then tries to maximize this term, by outputting values near 0. Conversely, the generator tries to minimize the term by making $D(G(x))$ close to 1, which means the fake data sample successfully fooled the discriminator.

This objective function defines the goals of each of the adversarial networks. In most cases, D is optimized more often than G, in order to keep the value near the optimal solution. Through multiple training loops, both D and G get closer to maximizing and minimizing their respective targets, training the network to create convincing data.

GAN models are typically implemented through neural networks. Most networks are either feed-forward or convolutional models. While GAN models can be useful, they can suffer from mode collapse, training instability, and susceptibility to overfitting that can create poor datasets. An alternative to GAN models that mitigates these downsides are WGAN models.

[1] The expectation $\mathbb{E}$ of a random variable gives a weighted average where each possible outcome is weighted by its possibility. Mathematically, the expectation of a set X with probabilities p is given by $\mathbb{E}[X] = \sum_{i=1}^{\infty} x_i p_i$

2.1 WGAN Framework

While basic GAN models can work well, there are several issues that can produce unreliable results. A large issue traditional GAN models face is mode collapse. Mode collapse is when the generator begins producing a limited variety of outputs. For example, a generator set to output columns of a mix of 1's and 0's may start outputting columnns of exclusively 1's or exclusively 0's. The failure of generators to create realistic data is a significant challenge for synthetic data models.

One method to avoid mode collapse is to switch from a traditional GAN to a Wasserstein GAN (WGAN). These models use Wasserstein distance, otherwise known as Earth Mover's Distance (EMD), as the loss function for the model. Wasserstein distance measures the work needed to transform one probability distribution into another [5]. The formal defition of Wasserstein Distance $W_1(P, Q)$ is

$$W_1(P, Q) = \int_{-\infty}^{\infty} |F_p(x) - F_q(x)| dx$$

.

For discrete probabilities, this can be changed to

$$W_1(P, Q) = \sum_i |F_p(x_i) - F_q(x_i)|$$

Intuitively, you can think of the Wasserstein Distance as a measure of how similar two probability distributions are. Imagine that there are two piles of dirt, and you need to move one pile until it's identical to the other. The Wasserstein distance tells you how much work you'll need to do to lift and move one of the piles until it's the same. If you think of the dirt piles as probability distributions (bell curves), you can see the Wasserstein distance gives you the work done to reshape one distribution to match the other. This is a good measure of how similar two probability distributions are, as the more similar the distribution, the less work needs to be done.

WGAN models use a modified Wasserstein distance as the loss metric. To make this work, WGAN models replace the discriminator with a *critic*. Instead of outputting a value representing the probability of a data sample being real or not, the critic assigns a score to each data sample. For x sampled from the real dataset P_r and y sampled from the generated date P_g, the score is given by

$$L_D = \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{y \sim P_g}[f(y)]$$

While not exactly an average, the expectation $\mathbb{E}$ can be approximated using tf.reduce_mean(). Averaging the distributions this way produces a close estimate to the Wasserstein loss, at a much more computationally efficient cost. To stabilize the estimation process, a gradient penalty is introduced.

2.2 Gradient Penalty

There is an extension of the WGAN framework that includes a gradient penalty, often called WGAN-GP. The gradient penalty was introduced as a way to stabilize training and resolve errors caused by the earlier WGAN models.

In order for the Wasserstein distance to be defined properly and ensure the optimization algorithm works correctly, the critic must satisfy a 1-Lipschitz Condition. In mathematical analysis, a Lipschitz-continuous function f is defined as $|f(x) - f(y)| \leq L \cdot |x - y|$ for all x, y and for some fixed L. The constant L is known

as the Lipschitz constant. For a 1-Lipschitz condition, $L = 1$, so our function must satisfy $|f(x) - f(y)| \leq |x - y|$. In practice, this means the difference between two output values in a function is at most as large as the distance between the input values. This ensures that small changes in an input don't rapidly change the value of the function, which can lead to uncontrolled gradients. Rapid, uncontrolled gradient changes cause errors in training, and can lead to a poorly functioning model. To enforce 1-Lipschitz continuity, the gradient penalty is defined as

$$L_{GP} = \lambda \mathbb{E}_{\hat{x}} \left[\left(||\nabla_{\hat{x}} D(\hat{x})|| \right]_2 - 1)^2 \right]$$

The term $||\nabla_{\hat{x}} D(\hat{x})||_2$ is the L_2 norm of the gradient [6]. Gradient norms provide the magnitude of the gradient vector, or the rate of change of the function in the direction of the gradient. For example, a large norm tells you a function is increasing rapidly, while a small norm tells you it changing slowly or near a local extremum. The L_2 norm is defined as

$$||\nabla_{\hat{x}} D(\hat{x})||_2 = \sqrt{\sum_{i=1}^{n} \left(\frac{\partial D}{\partial x_i} \right)^2}$$

In essence, the WGAN gradient penalty measures the difference of the magnitude of the gradient and 1, and penalizes gradients that deviate from 1. The parameter λ is a hyperparameter that controls how strong of a penalty there is for solutions that deviate too far from 1. By forcing gradient norms to be close to 1, the critic avoids sharp gradients, ensuring smooth and stable training.

3 Case Study: Customer Churn Detection

The initial problem that sparked my research into GAN networks was customer churn detection. The factors that indicate declining customer buying habits are typically very complex, requiring lots of high quality, labeled training samples to classify the risk of customer churn. Unfortunately, the data sets that I was utilizing were not very high-quality. While the model for churn detection had an accuracy of 94%, it was not robust to outliers and would regularly misclassify what should have been easy samples.

To get around this, I initially tried making synthetic datasets to model customer purchase data using Numpy. While somewhat accurate, they did not capture the volatility and long term trends that the real customer data did. After hitting dead end after dead end with synthetic data creation, I turned to GAN models to create high-quality data.

GAN models are widely used to create synthetic datasets. Since the end goal of a GAN network is to differentiate fake data from real data, a well-trained model should be able to generate fake data that is nearly indistinguishable from real data. This positions GAN models as strong methods for developing synthetic data. When GAN-generated synthetic data is blended into real data, the model's accuracy significantly increased. Researchers for the IEEE were able to increase the sensitivity and specificity of a biomedical imaging model from 78.6% and 88.4% to 85.7% and 92.4% respectively, only by including GAN generated data in training sets [7].

3.1 Why WGAN-GP?

WGAN was chosen for this project due to its higher data quality. When comparing generated data to the real data set, the WGAN model was much more realistic. This likely was due to the issue of mode collapse. Switching from a traditional GAN to WGAN fixed this issue, and produced much more realistic data. Including the gradient penalty ensured smooth and stable training, which was a necessity since the training data was volatile time series data.

The WGAN-GP was implemented in Python using the TensorFlow library. The generator and discriminator were constructed using a mix of LSTM and Conv1D layers. One-dimensional convolutional layers (Conv1D) utilize sliding kernels to extract short-term trends from time series data. Opposing the Conv1D layers are Long Short Term Memory (LSTM) neurons, which use memory cells to learn information about longer-term trends. An interesting feature of LSTM neurons is the forget gate, which allows the neuron to discard irrelevant information, making the neuron more adaptive as it learns. Since the WGAN was attempting to create time series data, the combination of these two neuron types allowed for more accurate datasets.

In order to prepare data to be used in the GAN, it was first scaled to an appropriate range. Scaling data improves the quality of predictions through a few pathways. For models that are reliant on gradient manipulation (such as neural networks), convergence is much faster for scaled data than unscaled. This is due to all data being scaled to the same range, preventing outliers from dominating updates. For the GAN, data was scaled using the MinMaxScaler from scikit.learn. Data was scaled to $(-1, 1)$, which provides several advantages to the model.

Once data was scaled, the generator and critic networks were defined. The WGAN-GP framework is given below.

Layer	WGAN Generator	WGAN Critic
Input Layer	Dense	Conv1D
Hidden Layer 1	Conv1D(tanh)	Leaky ReLU
Hidden Layer 2	Conv1D(tanh)	Conv1D
Hidden Layer 3	Conv1D(tanh)	Leaky ReLU
Hidden Layer 4	Batch Normalization	Conv1D
Hidden Layer 5	Dense	Leaky ReLU
Hidden Layer 6	LSTM(tanh)	Flatten
Hidden Layer 7	LSTM(tanh)	
Hidden Layer 8	LSTM(tanh)	
Output Layer	Dense	Dense

Table 1: Layer Structures for WGAN Generator and Critic

Activation functions transform inputs to be passed on to the next layer, introducing non-linearity between layers. Similar to how neurons in our brains decide how to process signals, activation functions help a neuron decide whether it is in an "activated" state or not. For the generator, the function $tanh$ was used. The function is defined as $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. This activation was chosen due to its output being constrained to $-1, 1$, the same range that the data is scaled. Since there are also negative values present in the dataset, it was necessary to have an activation function that allowed for negative values.

The next step was to develop the critic. Similar to the generator, the critic used a mix of Conv1D and LSTM layers to build its predictions. The critic model is

given above in Table 1. The critic network is notably simpler than the generator network. This is due to the fact that the critic is updated multiple times per update of the generator. This helps ensure that the generator is able to learn the underlying trends in real data, while not overpowering the critic.

Leaky ReLU is utilized in the critic network to ensure that training is stable. ReLU (Rectified Linear Unit) activations are the most common activation used for deep learning networks. While traditional ReLU neurons are the usual standard, they have a few problems that can hamper the ability of a network to perform optimally. Since the output of ReLU is constrained to to $(0, x)$, neurons that feed in negative values can "die" during training, since any negative value gets set to 0. This means certain parts of the network will become inactive during training, causing issues with the final network.

ReLU is defined as $\text{ReLU}(x) = \max(0, x)$. Leaky ReLU uses a slightly different piecewise formula:

$$\text{LeakyReLU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

where α is typically a small value. By allowing the activation function to keep negative values, the issue of dying neurons is avoided. This ensures the gradient flows smoothly for deeper networks, which helps stabilize training and ensures the critic is able to make good predictions.

For WGAN training, the ideal loss values are a negative generator loss and a critic loss of about -1. These values indicate that the generator is able to successfully fool the critic, while the critic is classifying data correctly without being overconfident. If the generator loss gets too large, it suggests that the generator isn't making random data samples. Conversely, too large of a negative value for the critic indicates the critic is overconfident in its predictions, causing the generator to make poor samples. In order to balance the two networks, a well-defined training loop is necessary.

4 Training Loop

When fully implemented, the training algorithm for a WGAN-GP model is given below. To compute the gradient penalty, interpolated data is created. Interpolated data samples are values that are estimated using known data points; in essence, interpolated data samples are fake data that is estimated using known trends in real data. Interpolated samples are used for data smoothing and have common applications in time series analysis, image processing and pre-processing data for machine learning. The training loop for a WGAN-GP model is described below.

Algorithm 1 Training Algorithm for WGAN-GP

Input: Learning rates α_D, α_G, batch size m, number of discriminator iterations per generator update n_D, gradient penalty coefficient λ

Initialize: Critic (discriminator) D and generator G with parameters θ_D and θ_G

for number of training iterations **do**

for n_D steps **do**

Sample batch of real data $\{y_1, y_2, \ldots, y_m\} \sim p_{\text{data}}$

Sample batch of random noise $\{z_1, z_2, \ldots, z_m\} \sim p_z$

Generate fake samples $\{x_1, x_2, ..., x_m\}$ where $x_i = G(z_i)$

Generate a random scalar ϵ in $(0, 1)$: $\epsilon \sim U(0, 1)$

Compute interpolated samples: $\hat{x}_i = \epsilon y_i + (1 - \epsilon)x_i$

Compute critic loss:

$$L_D = \frac{1}{m}\sum_{i=1}^{m}[D(x_i) - D(y_i)] + \lambda\left(\frac{1}{m}\sum_{i=1}^{m}\left(\|\nabla_{\hat{x}_i} D(\hat{x}_i)\|_2 - 1\right)^2\right)$$

Update critic using gradient descent: $\theta_D \leftarrow \theta_D - \alpha_D \nabla_{\theta_D} L_D$

end for

Sample batch of random noise $\{z_1, z_2, \ldots, z_m\} \sim p_z$

Generate fake samples: $x_i = G(z_i)$ for $i = 1, \ldots, m$

Compute generator loss:

$$L_G = -\frac{1}{m}\sum_{i=1}^{m} D(G(z_i))$$

Update generator using gradient descent: $\theta_G \leftarrow \theta_G - \alpha_G \nabla_{\theta_G} L_G$

end for

5 Results

The WGAN-GP model successfully generated synthetic customer data that improved the accuracy of a separate neural network from 93% to 98%. This improvement was primarily due to the model's ability to generate realistic outlier cases, which the previous training set lacked.

However, this accuracy gain came with a computational tradeoff. Traditional synthetic data generation (using NumPy and Pandas) produced 100 samples in 30 minutes, whereas WGAN-GP required 50 minutes to generate just 25 samples, making it approximately seven times slower. During testing, the final sizes of the data sets were 4,000 entries for the Numpy data set and 900 for WGAN-GP and Numpy together. When computed, the time needed to create the full data set using Numpy was about 1,200 minutes, or about 20 hours. The time needed to create the data set using WGAN-GP and Numpy together was about 7.33 hours. Although WGAN-GP takes more time to train than pure Numpy, the overall time required to create the data sets was 63.35% shorter.

Despite this, WGAN-GP data proved more efficient in reducing overall training needs. Achieving 93% accuracy with traditional synthetic data required 4,000 samples, while a mix of 100 WGAN-GP samples and 800 traditional samples achieved 98% accuracy. Utilizing WGAN-GP, the accuracy was improved by 5% while reducing the necessary training data by over 75% and training time by 63.35%. This

suggests that although GAN-based models are computationally expensive, they can reduce total training and data creation time by improving data quality and reducing the number of required samples.

6 How Businesses Drive Profitability with GAN

While GAN models have been widely implemented in areas like healthcare, private industry can also benefit significantly from the increase in data quality GAN models provide. By creating synthetic high-quality data that mirrors real data, businesses are able to expand their existing high-quality data sets. In a survey by Deloitte, 67% of executives say they are not comfortable using their existing data due to quality concerns [1]. By implementing GAN models, high quality data can be created to put these fears to rest.

Current research shows a clear link between data quality and model performance. A study conducted on a large language model found that training on only 30% of a data set yielded similar, and even better results in some cases, compared to the full training set. The reason this happened was due to the smaller data set containing higher quality data. Since training time is directly related to the size of the training set, this means a better-trained model can be produced in about 70% less time by increasing data quality.

My own findings with WGAN-GP confirms this research. By incorporating WGAN-GP samples into synthetic data sets, the training data size was reduced by 75% and the total data creation time was reduced by 63.35%, while accuracy improved by 5%. As the cost of data storage and training AI models continue to increase, any reduction in the needed storage space and cloud resources save costs. Currently, the cost for a cloud-hosted TPU (Tensor Processing Unit) for deep learning averages between $5-8 per hour. When you consider that some models take days to weeks to train, and may need to be trained dozens of times before production, the costs for a single AI model can exceed tens or even hundreds of thousands of dollars before being put into production. A reduction of over 60% for training time is massive savings for AI-savvy businesses.

While AI adoption leads to increased efficiency and profitability for many industries, there are often roadblocks along the journey. One of the largest is the quality of data available. This article has hopefully highlighted how GAN models can improve data quality while simultaneously reducing the overhead costs of AI, including data storage and training costs. As businesses look to stay ahead in the rapidly evolving AI landscape, embracing GAN models to enhance data quality can be the key to unlocking significant cost savings and accelerating the adoption of AI solutions.

References

[1] Leavy, E. (2023, August 29). Data Quality Crisis: New survey reveals 77% of organizations have quality issues. AI, Data & Analytics Network. https://www.aidataanalytics.network/data-governance/articles/data-quality-crisis-new-survey-reveals-77-of-organizations-have-quality-issues: :text=Data%20Quality%20Crisis%3A%20New%20Survey,of %20Organizations%20Have%20Quality%20Issues

[2] Niu, Z.; Yu, K.; Wu, X. LSTM-Based VAE-GAN for Time-Series Anomaly Detection. Sensors 2020, 20, 3738. https://doi.org/10.3390/s20133738

[3] Abedi, M., Hempel, L., Sadeghi, S., & Kirsten, T. (2022). Gan-based approaches for generating structured data in the medical domain. Applied Sciences, 12(14), 7075. https://doi.org/10.3390/app12147075

[4] Saifur Rahman, Shantanu Pal, Shubh Mittal, Tisha Chawla, Chandan Karmakar, SYN-GAN: A robust intrusion detection system using GAN-based synthetic data for IoT security, Internet of Things, Volume 26, 2024, 101212, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2024.101212. (https://www.sciencedirect.com/science/article/pii/S2542660524001537)

[5] Chilamkurthy, K. (2020, October 23). Wasserstein distance, contraction mapping, and modern RL theory. Medium. https://kowshikchilamkurthy.medium.com/wasserstein-distance-contraction-mapping-and-modern-rl-theory-93ef740ae867

[6] Sankar, A. (2025, March 5). Demystified: Wasserstein Gan with gradient penalty. Towards Data Science. https://towardsdatascience.com/demystified-wasserstein-gan-with-gradient-penalty-ba5e9b905ead/

[7] M. Frid-Adar, E. Klang, M. Amitai, J. Goldberger and H. Greenspan, "Synthetic data augmentation using GAN for improved liver lesion classification," 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018), Washington, DC, USA, 2018, pp. 289-293, doi: 10.1109/ISBI.2018.8363576. keywords: Lesions;Liver;Gallium nitride;Training;Medical diagnostic imaging;Task analysis;Image synthesis;data augmentation;generative adversarial network;liver lesions;lesion classification,

[8] Randieri, C. (2024, October 21). The importance of data quality: Metrics that drive business success. Forbes. https://www.forbes.com/councils/forbestechcouncil/2024/10/21/the-importance-of-data-quality-metrics-that-drive-business-success/