Generating Multi-type Temporal Sequences to Mitigate Class-imbalanced Problem

Lun Jiang*, Nima Salehi Sadghiani* (✉), Zhuo Tao*, and Andrew Cohen

Unity, San Francisco CA 94103, USA
{lun,nimas,zhuo,andrew.cohen}@unity3d.com

Abstract. From the ad network standpoint, a user's activity is a multi-type sequence of temporal events consisting of event types and time intervals. Understanding user patterns in ad networks has received increasing attention from the machine learning community. Particularly, the problems of fraud detection, Conversion Rate (CVR), and Click-Through Rate (CTR) prediction are of interest. However, the class imbalance between major and minor classes in these tasks can bias a machine learning model leading to poor performance. This study proposes using two multi-type (continuous and discrete) training approaches for GANs to deal with the limitations of traditional GANs in passing the gradient updates for discrete tokens. First, we used the Reinforcement Learning (RL)-based training approach and then, an approximation of the multinomial distribution parameterized in terms of the softmax function (Gumble-Softmax). Our extensive experiments based on synthetic data have shown the trained generator can generate sequences with desired properties measured by multiple criteria.

Keywords: Multi-type sequences · Temporal events · Generative adversarial network · Reinforcement learning

1 Introduction

Game developers can monetize their games by selling in-game ad placements to advertisers. Ads can be integrated in multiple ways such as a banner in the background or commercials during breaks (when a specific part of the game is completed). There are four main elements in the game advertising ecosystem: publishers or developers, advertisers, advertising networks, and users [21]. Game advertising networks connect advertisers with game developers and serve billions of ads to user devices, triggering enormous ad events. For example, Unity Ads reports 22.9B+ monthly global ad impressions, reaching 2B+ monthly active end-users worldwide [1].

An ad event is a user interaction e.g. request, start, view, click, and install. Each type stands for one specific kind of ad-related user action happening at a specific time. A complete ad life cycle consists of a temporal sequence of ad

* Authors contributed equally.
[1] https://www.businesswire.com/news/home/20201013005191/en/

events, each of which is a tuple of event types with corresponding time intervals. Click and install are two kinds of ad events commonly associated with ad revenue. Pay-Per-Click [17] and Pay-Per-Install [27] are the most widely used advertising models for pricing.

Unlike traditional advertising, online advertising offers services that link user interactions to conversions or clicks. Due to this, predicting a user's probability of clicking or conversion rate has become one of the most important problems in online advertising [4]. Predicting Conversion Rate (CVR) and Click-Through Rate (CTR) are usually treated as supervised learning problems [7]. For example, in CTR prediction, the labels are click/not-click an ad for every user. The sequence of events before a click/not-click response are used as features of the supervised learning model.

Unfortunately, as advertisers allocate more of their budget into this ecosystem, there is more incentive to abuse the advertising networks and defraud advertisers of their money [22]. Fraudulent ad activity aimed at generating illegitimate ad revenue or unearned benefits are one of the major threats to online advertising models. Common types of fraudulent activities include fake impressions [14], click bots [13, 19], or click farms [24].

Given the massive ad activity data in-game advertising networks, machine learning-based approaches have become popular in the industry. However, it is not a straightforward task to train machine learning models directly on the sequences collected from ad activities [5].

The primary issue in these problems is class imbalance. By definition, the ratio of typical user behavior to anomalous will heavily favor typical. For example, the CVR can be as low as 0.01% for game ads. Similarly, most ad traffic is non-fraudulent, and data labeling by human experts is time-consuming. In these scenarios, label sparsity leads to low availability of labeled sequences for the minor class. Simply oversampling the minority class can cause significant overfitting, while undersampling the majority may lead to information loss and yield a tiny training dataset [1]. In this study, we present a novel method to generate synthetic data to mitigate class imbalance.

The main contributions of our work can be summarized as follows:

1. A novel reinforcement learning formulation that trains a generator to generate multi-type temporal sequences with non-uniform time intervals.
2. A novel training method for sequence GAN that uses a critic network.
3. A new application for event-based sequence GAN in game advertising.

2 Related Work

Generative Adversarial Networks (GANs) [11] have drawn significant attention as a framework for training generative models capable of producing synthetic data with desired structures and properties [18]. It was proposed to use GANs to generate data that mimics training data as an augmented oversampling method with an application in credit card fraud. The generated data is used to assist the classification of credit card fraud [1].

2.1 GAN for Sequence Data

Despite the remarkable success of GANs in generating synthetic data, very few studies focus on generating sequential data. This is due to additional challenges in generating temporally dependent samples. Recurrent Neural Network (RNN) solutions are state-of-the-art in modeling sequential data. Recurrent Conditional GAN (RCGAN) generates real-valued multi-dimensional time series and then uses the generated series for supervised training [10]. The time series data in their study were physiological signals sampled at specific fixed frequencies. However, ad event data has higher complexity due to non-uniform time intervals and discrete event types and thus can not be modeled as wave signals. In ad event sequences, two events with a short time interval tend to be more correlated than events with larger time intervals.

A GAN-based generative model for DNA along with an activation maximization technique for DNA sequence data is proposed by [18]. Their experiments have shown that these generative techniques can learn the important structure from DNA sequences and can be used to design new DNA sequences with desired properties. Similarly to the previous study, their focus is on fixed interval sequences.

The Long Short-Term Memory (LSTM)-Autoencoder is used to encode the benign users into a latent space [30]. They proposed using One-Class Adversarial Network (OCAN) for the training process of the GAN model. In their training framework, the discriminator is trained to be a classifier for distinguishing benign users, and the generator produces samples that are complementary to the representations of benign users.

2.2 RL for GANs with Sequences of Discrete Tokens

When generating continuous outputs, gradient updates can be passed from the discriminator to the generator. However, for discrete outputs, this is not straightforward due to a lack of differentiability. The issue of training GAN models to generate sequences of discrete tokens is addressed in [28]. They proposed a sequence generation framework called SeqGAN that models the data generator as a stochastic policy learned via Reinforcement Learning (RL) [26]. SeqGAN learns a policy using the vanilla policy gradient and Monte Carlo (MC) rollouts to approximate the advantage. MC rollouts are a computationally expensive process in the training loop. Moreover, SeqGAN is limited to discrete token generation. In our work, we propose a modified version of SeqGAN that can generate both discrete tokens and continuous time-intervals. Additionally, to efficiently train the policy network, we employ a Critic network to approximate the return given a partially generated sequence to speed up the training process. This approach also brings the potential to use a trained Critic network for early fraud detection from partial sequences.

An application of SeqGAN in recommendation systems is presented in [29]. The paper solves the slow convergence and unstable RL training by using the Actor-Critic algorithm instead of MC roll-outs. Their generator model produces

the entire recommended sequences given the interaction history while the discriminator learns to maximize the score of ground-truth and minimize the score of generated sequences. In each step, the generator G generates a token by top-k beam search based on the model distribution. In our work, we directly sample from the distribution of the output probabilities of the tokens. While our methodologies are close, we are aiming for different goals. We optimize the generated data to solve the sample imbalance problem while they optimize for better recommendations. Therefore, different evaluation metrics are needed. Our methodologies also differ in the training strategy. For example, we used a Critic network as the baseline, whereas they used Temporal-Difference bootstrap targets. They pre-trained the discriminator on the generated data to reduce the exposure bias, while we pre-trained the discriminator on the actual training data for improving the metrics we use in our experiments. More importantly, they do not include time intervals as an attribute in their model while we have time intervals in our models.

The idea of using SeqGan to adversarially learn the output sequences while optimizing towards chemical metrics with the algorithm REINFORCE [26] is proposed in [12]. They have shown that it is often advantageous to guide the generative model towards some desirable characteristics, while ensuring that the samples resemble the initial distribution.

2.3 Gumbel-Softmax Distribution for GANs with Sequences of Discrete Tokens

The Gumbel-Softmax distribution is proposed in [20] to address the limitation of GANs for generating sequences of discrete tokens. The Gumbel-Softmax is a continuous approximation to a multinomial distribution parameterized over a softmax function. This approximation is differentiable thus enabling backpropagation through an approximation of a discrete sampling procedure. A temperature parameter can be used to controll the degree of approximation [16]. When the temperature is lower, the approximation is closer to the one hot distribution; when it is higher, the approximation is closer to a uniform distribution.

Another application of Gumbel-Softmax distributions is proposed in [6] for generating small molecular graphs.

3 Methodology

In this section, we introduce a new methodology to generate multi-type sequences using GAN, which can be trained by using RL and Gumbel-Softmax reparametrization.

3.1 Definitions

The sequence of an ad event with length L is composed of two sub-sequences, the sub-sequence of event types $\boldsymbol{x}$ and the sub-sequence of time stamps. First, we

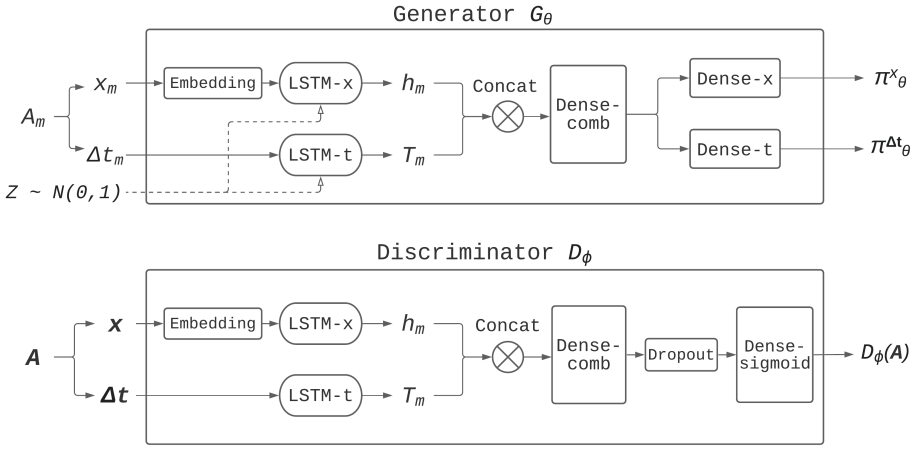

Fig. 1. Architecture of the Generator and the Discriminator.

transform the time stamps $\boldsymbol{t}$ into time intervals $\boldsymbol{\Delta t}$ and $\Delta t_m = t_m - t_{m-1}, \forall m \in [1, L]$, and $\Delta t_1 = t_1 - 0$. Then, we combine the event types and time intervals into a joint multi-type sequence $\boldsymbol{A}$:

$$\boldsymbol{A} = \boldsymbol{A}_{1:L} = \{(x_1, \Delta t_1), (x_2, \Delta t_2), \ldots, \Delta(x_m, \Delta t_m), \ldots, (x_L, \Delta t_L)\}$$

where a bold $\boldsymbol{A}_{1:m}$ denotes a partial sequence from step 1 to step m, and a non-bold $A_m = (x_m, \Delta t_m)$ denotes a single pair in the sequence.

3.2 RL and Policy improvement to train GAN

We implemented a modified version of SeqGAN model to generate multi-type temporal sequences. The architecture is shown in Fig. 1.

The sequence generation process of our generator G can be modeled as a sequential decision process in RL. h_m and T_m are the hidden states of LSTM cells, and $Z \sim \mathcal{N}(0, 1)$ is the normal noise used to initialize h_m and T_m at the beginning of each generation process.

From the perspective of RL, at each step m, we define the state S_m as the partial sequence $\boldsymbol{A}_{1:m}$, a.k.a,

$$S_m = \boldsymbol{A}_{1:m} \tag{1}$$

During the generation process, at each step m, a new pair

$$A_{m+1} = (x_{m+1}, \Delta t_{m+1}) \tag{2}$$

is appended to the current partial sequence $\boldsymbol{A}_{1:m}$ to formulate a new partial sequence $\boldsymbol{A}_{1:m+1}$, and thus transit to a new state S_{m+1}, based on the definition of state in (1). This process repeats step by step, until a complete sequence $\boldsymbol{A}$ of length L described in (3.1) is fully constructed.

To make decisions in this sequence generation process, we employ a hybrid policy to represent action spaces with both continuous and discrete dimensions (similar to the idea in [23]). This policy is designed to choose discrete event types and continuous time intervals, assuming their action spaces are independent. Then we use a categorical distribution and a Gaussian distribution to model the policy distributions for the event types and the time intervals respectively. So the hybrid generator policy can be defined as:

$$\begin{aligned} G_\theta(a_m|S_m) =& \pi_\theta^x(a_m^x|S_m) \cdot \pi_\theta^{\Delta t}(a_m^{\Delta t}|S_m) \\ =& Cat(x|\alpha_\theta(S_m)) \cdot \mathcal{N}(\Delta t|\mu_\theta(S_m), \sigma_\theta^2(S_m)) \end{aligned} \tag{3}$$

where $x \in \boldsymbol{K}, \Delta t \in R_{\geq 0}$. $\boldsymbol{K}$ is the set of all event types. Then an action a_m is taken at step m to sample the next event type x_{m+1} and the next time interval Δt_{m+1} given the hybrid policy (3). So the action has discrete part and the continuous part sampled independently:

$$a_m = \{a_m^x, a_m^{\Delta t}\} \tag{4}$$

$$a_m^x = x_{m+1} \sim Cat(x|\alpha_\theta(S_m)) \tag{5}$$

$$a_m^{\Delta t} = \Delta t_{m+1} \sim \mathcal{N}(\Delta t|\mu_\theta(S_m), \sigma_\theta^2(S_m)) \tag{6}$$

where a_m^x is the action to find the next event type x_{m+1} and $a_m^{\Delta t}$ is the action to find the next time interval Δt_{m+1}.

When generating a new event type and time interval at each step, we follow the generator policy and sample from categorical and Gaussian distributions independently and concatenate them to obtain the action vector a_m, then append them to the current partial sequence $\boldsymbol{A}_{1:m}$ to obtain a new partial sequence $\boldsymbol{A}_{1:m+1}$. Once a complete sequence of length L has been generated, we pass the sequence $\boldsymbol{A}$ to the Discriminator D which predicts the probability of the sequence to be real against fake:

$$D_\phi(\boldsymbol{A}) = Pr(Y = 1|\boldsymbol{A}; \phi) \tag{7}$$

The feedback from D can be used to train G to generate sequences similar to real training data to deceive D. Because the discrete data is not differentiable, gradients can not passed back to generator like in image-base GANs.

The original SeqGAN training uses Policy Gradient method with MC roll-out to optimize the policy.[28] In order to reduce variance in the optimization process, SeqGAN runs the roll-out policy starting from current state till the end of the sequence for multiple times to get the mean return. Here we use an Actor-Critic method with a Critic network instead of MC roll-out to estimate the value of any state, which is computationally more efficient.[2]

The critic network models a state-dependent value $\hat{V}_\psi^{G_\theta}(S_m)$ for a partially generated sequence $\boldsymbol{A}_{1:m}$ under policy G_θ. The output of the critic is defined as the expected future return for the current state $S_m = \boldsymbol{A}_{1:m}$, which will be given by the discriminator D when a complete sequence $\boldsymbol{A}$ is generated.

$$\hat{V}_\psi^{G_\theta}(S_m) = \mathbb{E}_{\boldsymbol{A}_{m+1:L} \sim G_\theta(S_m)}[D_\phi(\boldsymbol{A})] \tag{8}$$

The parameters in the critic value function $\hat{V}_{\psi}^{G_\theta}(S_m)$ are updated during training by minimizing the mean squared error between the true return $D_\phi(\boldsymbol{A})$ and the critic value:

$$J(\psi) = \mathbb{E}[(D_\phi(\boldsymbol{A}) - \hat{V}_{\psi}^{G_\theta}(S_m))^2] \tag{9}$$

The difference between them, $D_\phi(\boldsymbol{A}) - \hat{V}_{\psi}^{G_\theta}(S_m)$, is named the advantage function, which can be used in G training and helps to reduce variance.

The goal of G training is to choose actions based on a policy that maximizes expected return. The object function of G follows Policy Gradient method [26] which can be derived as:

$$\nabla_\theta J(\theta) = \sum_{m=0}^{L-1} \mathbb{E}_{a_m \sim G_\theta(a_m|S_m)}[\nabla_\theta \log G_\theta(a_m|S_m) \cdot (D_\phi(\boldsymbol{A}) - \hat{V}_{\psi}^{G_\theta}(S_m))] \tag{10}$$

Because of the independence assumption we made, the policy gradient term can be broken down and written into a categorical cross-entropy and a Gaussian log-likelihood as follows:

$$\begin{aligned}
&\nabla_\theta \log G_\theta(a_m|S_m) \\
=&\nabla_\theta[\log Cat(x = x_{m+1}|\alpha_\theta(S_m)) + \log \mathcal{N}(\Delta t = \Delta t_{m+1}|\mu_\theta(S_m), \sigma_\theta^2(S_m))] \\
=&\nabla_\theta[\mathbb{E}_{x \in \boldsymbol{K}} \mathbb{1}_x(x_{m+1}) \Pr(x = x_{m+1}) - \frac{(\Delta t_{m+1} - \mu_\theta(S_m))^2}{2\sigma_\theta^2(S_m)} - \frac{1}{2}\log(2\pi\sigma_\theta^2(S_m))]
\end{aligned} \tag{11}$$

The goal of D training to use distinguish generated sequences with true sequences from training data. D_ϕ is updated through minimizing binary cross-entropy loss. G and D alternatively in GAN training.

The training data are taken from the positive class $\boldsymbol{\Omega}^+$ of our synthetic Ad event dataset $\boldsymbol{\Omega}$, which are shown in the section 4.1.

Before GAN training, We pre-train G with Maximum Likelihood Estimation (MLE) self-regression on the sequences and pre-train D with binary classification for better convergence. Details about pre-training and GAN training The Pseudo code of the entire process is shown in Algorithm 1.

3.3 An Approximation with Gumbel-Softmax Distribution

Beside RL, we also tried to overcome the gradient updates problem for discrete token in GAN using Gumbel-Softmax reparametrization. We use the same generator G and discriminator D setups as described in section 3.2, except that the generator policy $G_\theta(a_m|S_m)$ is different from that in (3). For the continuous part, we no longer sample time intervals from a parametrized Normal distribution, but directly take G outputs as the next time interval.

$$a_m^{\Delta t} = \Delta t_{m+1} = \Delta t_\theta(S_m) \tag{12}$$

Algorithm 1 Sequence Generative Adversarial Nets Training with RL

Require: training dataset $\boldsymbol{\Omega}^{+}$, generator G_θ, discriminator D_ϕ, critic $\hat{V}_\psi^{G_\theta}$.

Initialize G_θ, D_ϕ, $\hat{V}_\psi^{G_\theta}$ with random weights θ, ϕ, ψ
Pre-train G_θ with MLE self-regression on $\boldsymbol{\Omega}^{+}$.
Generate fake dataset $\boldsymbol{\Omega}^{+fake}$ using pre-trained G_θ.
Pre-train D_ϕ via minimizing binary cross-entropy on $\boldsymbol{\Omega}^{+} \cup \boldsymbol{\Omega}^{+fake}$
repeat
    for G-steps **do**
        Generate a batch of fake sequences $\boldsymbol{A}^{fake} \sim G_\theta$
        Get true rewards $D_\phi(\boldsymbol{A})$ from discriminator
        for m in $1:L$ **do**
            $S_m \leftarrow \boldsymbol{A}_{1:m}^{fake}$
            $a_m \leftarrow (x_{m+1}, \Delta t_{m+1}) \in \boldsymbol{A}^{fake}$
            $\alpha_\theta(S_m), \mu_\theta(S_m), \sigma_\theta(S_m) \leftarrow G_\theta(S_m)$
            Compute policy gradient as shown in Eq. (11)
            Compute value estimate $\hat{V}_\psi^{G_\theta}(S_m))$ by Eq. (8)
            Compute the advantage $(D_\phi(\boldsymbol{A}) - \hat{V}_\psi^{G_\theta}(S_m))$
        Update critic param. ψ by minimizing Eq. (9)
        Update generator param. θ via Eq. (10)
    for D-steps **do**
        Generate a batch of sequences $\boldsymbol{A}^{fake} \sim G_\theta$
        Sample a batch of sequences $\boldsymbol{A}^{true}$ from $\boldsymbol{\Omega}^{+}$
        Train discriminator D_ϕ on $\boldsymbol{A}_{fake} \cup \boldsymbol{A}_{true}$ and update param. ϕ via minimizing binary cross-entropy
until terminate condition satisfied

For the discrete part, in the forward pass of training the generator G, we add a Gumbel noise to the probability distribution of event types at each step m, and use argmax operator to sample the next event type x_{m+1}:

$$a_m^x = x_{m+1} = \arg\max_i(\log(\alpha_\theta(S_m)_i) + g_i) \quad \text{for} \quad i = 1, \ldots, |\boldsymbol{K}| \tag{13}$$

where τ is the temperature and g is a random variable with a standard Gumbel distribution:

$$g = -\log(-\log(U)), \quad \text{where} \quad U \sim \text{Uniform}([0, 1]) \tag{14}$$

In the backward pass of G training, we reparametrize the categorical distribution using a Gumbel random variable g to create a differentiable approximation of the discrete representation of a_m^x to calculate gradients:

$$\Pr(a_m^x = x_i, x_i \in \boldsymbol{K}|S_m) = \frac{\exp\left((\log(\alpha_\theta(S_m)_i) + g_i)/\tau\right)}{\sum_{j=1}^{k} \exp\left((\log(\alpha_\theta(S_m)_j) + g_j)/\tau\right)} \quad \text{for} \quad i = 1, \ldots, |\boldsymbol{K}| \tag{15}$$

After the Gumbel-Softmax reparametrization, we can train the multi-type GAN with discrete event types using a similar approach in [20].

4 Data Experiments

Due to data privacy laws (e.g. GDPR [2], CCPA [3]), and to protect confidential details of the Unity Ads Exchange and Fraud Detection service, we opt not to use real-world ad events data in this study to avoid releasing user behavior patterns to the public. While anonymizing the real-world dataset can hide users' identities, it cannot disguise the users' behavior patterns and distributions. Fraudsters can easily employ bots to simulate the features of real users to bypass fraud detection systems, if given access to the real data.

Instead, we conduct our experiments on a synthetic dataset, which contains simplified data patterns we observed and abstracted from real-world ad events. The design philosophy is explained in Section 4.1. The synthetic dataset and code used to generate it are publicly available[4].

4.1 Synthetic Dataset

We define the synthetic dataset as $\boldsymbol{\Omega}$. There are 4 types of hypothetical ad events in $\boldsymbol{\Omega}$, shown as $\boldsymbol{K} = \{a, b, c, d\}$. Each sequence in the synthetic dataset $\boldsymbol{\Omega}$ has a uniform length $L = 20$. A step at m corresponds to a tuple of event type and

[2] General Data Protection Regulation

[3] California Consumer Privacy Act

[4] `https://github.com/project-basileus/multitype-sequence-generation-by-tlstm-gan`

time interval, $(x_m, \Delta t_m)$, where x_m is sampled uniformly from $\boldsymbol{K}$, and Δt_m is sampled from a Chi-Square distribution with the degree of freedom conditioned on x_m, i.e.:

$$x_m \sim \text{Uniform}\{a, b, c, d\} \quad \Delta t_m \sim \mathcal{X}^2(k), \quad k = \begin{cases} 10 & \text{if} \quad x_m = a \\ 20 & \text{if} \quad x_m = b \\ 40 & \text{if} \quad x_m = c \\ 80 & \text{if} \quad x_m = d \end{cases} \tag{16}$$

One example of a complete synthetic sequence is as below:

$$\begin{aligned} \boldsymbol{A}_{e.g.} = [&(a, 5), (a, 22), (b, 27), (c, 44), (c, 43), \\ &(d, 87), (b, 30), (c, 36), (d, 75), (c, 28), \\ &(a, 9), (b, 24), (a, 9), (c, 40), (b, 29), \\ &(c, 37), (a, 10), (b, 19), (c, 26), (b, 7)] \end{aligned}$$

There are two classes in $\boldsymbol{\Omega}$, the positive class $\boldsymbol{\Omega}^+$ and the negative class $\boldsymbol{\Omega}^-$. As the two classes can be highly imbalanced in real-world Ad events data (e.g. fraud/non-fraud, buyer/non-Buyer, conversion/non-conversion, etc.), the positive class is the minority in $\boldsymbol{\Omega}$, with a positive-to-negative ratio of $1 : 500$. A positive sequence has the following properties:

1. The time delay between any two consecutive events of the same event type is greater than or equal to 20.
2. Each d event is paired with one and only one previous c event. Each c event can be paired with at most one d event after it.
3. The time delay between any two paired c and d events is smaller than or equal to 200.

Sequences failing to have all 3 properties above are considered negative. The positive class $\boldsymbol{\Omega}^+$ is the training dataset. We train a GAN to generate data points from the minority class with the above properties. We will employ them as an oracle to evaluate the quality of GAN-generated sequences, as described in section 4.2.

The design philosophy of the synthetic dataset is to simulate real-world patterns with as much fidelity as possible while hiding real parameters to prevent reverse-engineering by fraudsters. Specifically, the hypothetical ad events $\{a, b, c, d\}$ mimic four typical real ad events: starts, views, clicks, and installs. Real-world time delay between ad events follows a long-tail distribution, while in the synthetic dataset, it is modeled with a Chi-Square distribution conditioned on the preceding event type. Moreover, the three properties of a positive sequence are also abstracted from real-world data patterns: property 1 detects high-frequency attacks; property 2 describes the ad attribution process between clicks and installs; property 3 checks the validity of an attribution window. Ad attribution refers to the process of determining the user actions that led to the desired outcome between the click of the ad and the conversion.

4.2 Evaluation Metric

In the last few years, several different evaluation metrics for GANs have been introduced in the literature. Among them, Fréchet Inception Distance (FID) [15] has been used extensively [8]. However, this only captures the numerical part of a sequence, but our sequences are multi-type containing both the discrete categorical part (event type) and the continuous numerical part (time interval). Thus, we propose using multiple metrics to measure the quality of generated sequences. We use Mean Absolute Deviation (MAD) to measure the discrete event types, and use FID to evaluate the continuous time intervals. In addition, we employ an oracle score based on the known properties in the training data to measure the similarity between generated sequences and the training data. The arrows (↑↓) show the improvement directions.

MAD ↓. We propose using MAD to evaluate the statistical dispersion between the categorical part (i.e., the event types) of the generated multi-type sequences and that of the training data. We use the training dataset $\boldsymbol{\Omega}^+$ as the comparison base, and then one-hot encode the event types of training sequences to calculate the medians at each step m. Median is known to be more robust to noise and fits our need to have categorical values as opposed to mean.

The MAD score of any batches of generated sequences $\boldsymbol{B}$ is computed as the mean absolute deviation of each sequence from the base medians, shown as below as MAD can be computed using:

$$MAD(\boldsymbol{B}) = \frac{1}{|\boldsymbol{B}|} \sum_{\boldsymbol{A} \in \boldsymbol{B}} \sum_{m=1}^{L} \left| x_m^{\boldsymbol{A}} - \tilde{E}_m(\boldsymbol{\Omega}^+) \right| \tag{17}$$

where $\boldsymbol{B}$ is a batch of generated sequences, $|\boldsymbol{B}|$ is the batch size, $\boldsymbol{A}$ is a sequence of length L in $\boldsymbol{B}$, $x_m^{\boldsymbol{A}}$ is the event type of step m in $\boldsymbol{A}$, $\tilde{E}_m(\boldsymbol{\Omega}^+)$ is the base median of the event types at step m across the training dataset $\boldsymbol{\Omega}^+$.

FID ↓. Similarly to MAD, we use FID to measure the distance between the numerical part (i.e., the time intervals) of the multi-type sequences and that of the training data. This score focuses on capturing certain desirable properties including the quality and diversity of the generated sequences. FID performs well in terms of robustness and computational efficiency [3]. The Fréchet distance between two Gaussians is defined as:

$$FID(x, g) = \left\| \mu_x - \mu_g \right\|_2^2 + Tr\left(\Sigma_x + \Sigma_g - 2\left(\Sigma_x \Sigma_g \right)^{\frac{1}{2}} \right) \tag{18}$$

where (μ_x, Σ_x) and $\left(\mu_g, \Sigma_g\right)$ are the means and covariances for the training and generated data distribution, respectively.

Oracle ↑. One of the most direct ways to measure the quality of a generated sequence is to check whether it has the known data properties of the positive class (described in section 4.1). For a batch of generated sequences, we calculate the percentage of sequences having all 3 properties of the positive class over all sequences, and then use this ratio as the oracle score. For example, for a data batch from the training dataset $\boldsymbol{\Omega}^+$, the oracle score is 1. The oracle score

Table 1. Oracle metrics calculated using $\boldsymbol{\Omega}^+$ as base

	Reinforcement Learning (RL)			Gumbel-Softmax (GS)		
Samp.	**MAD** $\downarrow$	**FID** $\downarrow$	**Oracle** $\uparrow$	**MAD** $\downarrow$	**FID** $\downarrow$	**Oracle** $\uparrow$
G0	0.8265	19892.4782	0.0015	0.7368	10045.2759	0.1477
G1	0.6622	**101.7972**	0.0820	0.6399	10455.6409	0.3600
G2	**0.2849**	6495.2955	**0.5407**	**0.5427**	**9111.5298**	**0.55875**

is a metric taking both the continuous and discrete part of a sequence into consideration.

4.3 Experiment Setup

We take 4000 samples from the $\boldsymbol{\Omega}^+$ dataset defined in section 4.1 for model training. As is described in Algorithm 1, we first pre-train G and D and then start GAN training from the pre-trained G and D. We define the following terms to describe the generator at different training phases:

- $G0$: Generator with initial random model parameters.
- $G1$: Generator pre-trained using MLE self-regression.
- $G2$: Generator after GAN training.

The ratio between G training steps and D training steps is set to $1:1$. Both G and D have the same batch size 256, and use the Adam optimizer with learning rate 10^{-4}.

During the pre-training and training processes, we evaluated the performance of the trained generator G after some steps. The trained generator was then used to generate a batch of data points and the batch evaluated according to the metrics defined in section 4.2.

To avoid mode collapse and convergence problems, we used several techniques including label smoothing and noisy labels [25] in GAN training. In RL training, we added entropy regularizers [9] to the reward for discrete token and continuous time interval generation to avoid over-fitting.

4.4 Experiment Results

Table 1 shows the evaluation metric values of the sequences generated by G at different phases of training. The MAD, FID score are calculated respectively using data sampled from $\boldsymbol{\Omega}^+$ as the base for the comparisons.

The curves of evaluation metrics during pre-training and training are shown in Figure 2 and Figure 3, respectively.

The results in table 1 demonstrate that the sequences generated by GAN-trained $G2$ have a significantly higher oracle score than that generated by the MLE pre-trained generator $G1$ and randomly initialized generator $G0$, for both RL and Gumbel-Softmax training. This indicates that the generator is able to

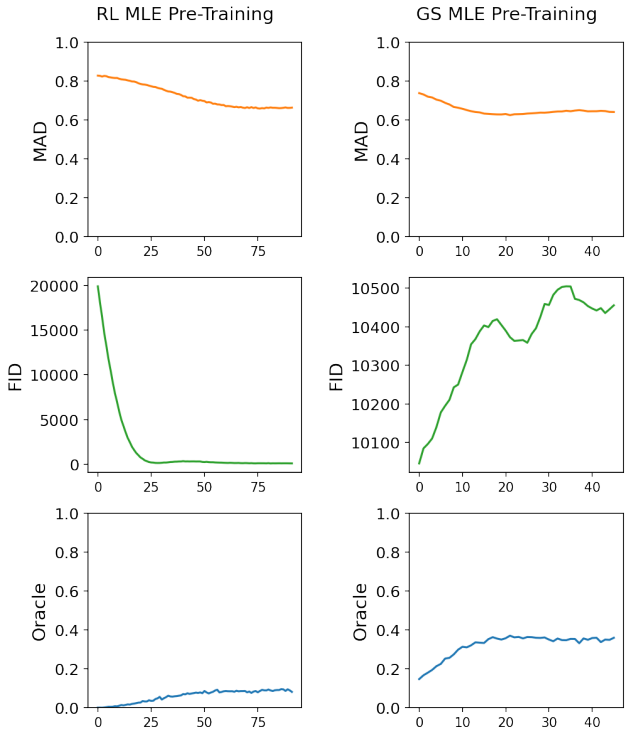

Fig. 2. Metrics of generated sequences over pre-training steps for Reinforcement Learning (RL) and Gumbel-Softmax (GS).

learn the intrinsic patterns and properties in the training data $\boldsymbol{\Omega}^+$, and is able to mimic these patterns to deceive the discriminator.

From the perspective of metric curves, we noticed that in the pre-training of RL, the FID score of the generator decayed sharply from around $20,000$ to around 100, while the the improvements of MAD score and oracle score were stalling. It suggested that the MLE training was over-fitting in learning the continuous distribution of the time interval Δt, while paying much less effort to learn the patterns in the discrete event type x, and the relationships and hidden connections between the continuous and the discrete parts.

Comparing the performance of RL and Gumbel-Softmax training approaches, we found that the RL approach converged faster in pre-training and training with smoother metrics curves, but it was vulnerable to over-fitting and Gaussian model collapsing. Meanwhile, the Gumbel-Softmax approach converged slower with more curve oscillations, but it was less prone to over-fitting, even with the entropy regularizers in reward.

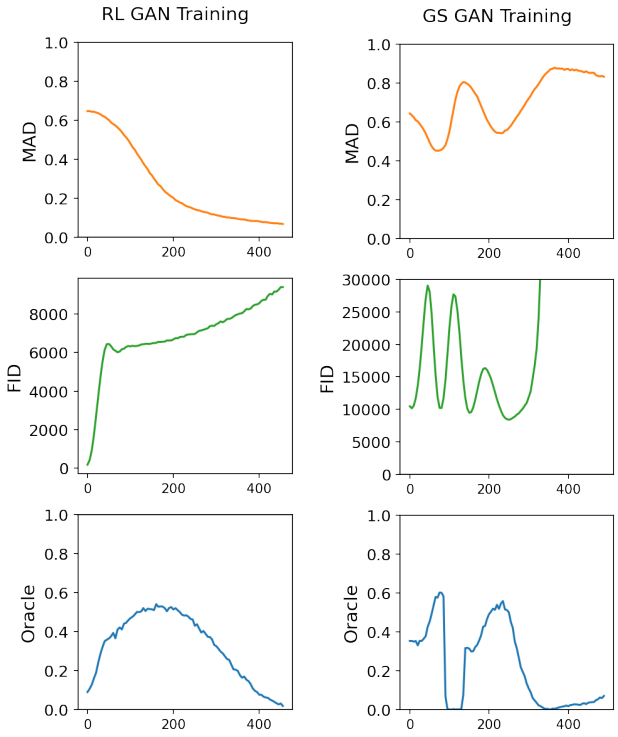

Fig. 3. Metrics of generated sequences over training steps for Reinforcement Learning (RL) and Gumbel-Softmax (GS).

5 Conclusions

In this paper, we have described, trained, and evaluated a novel methodology for generating artificial sequences with multi-type tokens. As this task poses new challenges, we have presented and compared the policy gradient (RL) and Gumbel-Softmax approaches for training a multi-type GAN. The generator proposed in this paper is capable of generating multi-type temporal sequences with non-uniform time intervals. We have also proposed using multiple criteria to measure the quality of the generated sequences. Experiments demonstrate that the generated multi-type sequences contain the desired properties.

Furthermore, we compared the performance of our generator for both RL and GS approaches with data from our carefully designed synthetic dataset. We concluded that the SeqGAN-trained generator has a higher performance compared to pre-trained generators using self-regression MLE, measured by multiple criteria including MAD, FID, oracle scores that are appropriate for evaluating multi-type sequences.

Acknowledgments

The authors would like to thank Unity for giving the opportunity to work on this project during Unity's HackWeek 2020.

References

1. Ba, H.: Improving detection of credit card fraudulent transactions using generative adversarial networks. arXiv preprint arXiv:1907.03355 (2019)
2. Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Naturalgradient actor-critic algorithms. Automatica (2007)
3. Borji, A.: Pros and cons of gan evaluation measures. Computer Vision and Image Understanding **179**, 41–65 (2019)
4. Chapelle, O., Manavoglu, E., Rosales, R.: Simple and scalable response prediction for display advertising. ACM Transactions on Intelligent Systems and Technology (TIST) **5**(4), 1–34 (2014)
5. Choi, J.A., Lim, K.: Identifying machine learning techniques for classification of target advertising. ICT Express (2020)
6. De Cao, N., Kipf, T.: Molgan: An implicit generative model for small molecular graphs. arXiv preprint arXiv:1805.11973 (2018)
7. Deng, C., Wang, H., Tan, Q., Xu, J., Gai, K.: Calibrating user response predictions in online advertising. In: Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part IV. pp. 208–223. Springer International Publishing (2021)
8. DeVries, T., Romero, A., Pineda, L., Taylor, G.W., Drozdzal, M.: On the evaluation of conditional gans. arXiv preprint arXiv:1907.08175 (2019)
9. Dieng, A.B., Ruiz, F.J., Blei, D.M., Titsias, M.K.: Prescribed generative adversarial networks. arXiv preprint arXiv:1910.04302 (2019)
10. Esteban, C., Hyland, S.L., Rätsch, G.: Real-valued (medical) time series generation with recurrent conditional gans. arXiv preprint arXiv:1706.02633 (2017)
11. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
12. Guimaraes, G.L., Sanchez-Lengeling, B., Outeiral, C., Farias, P.L.C., Aspuru-Guzik, A.: Objective-reinforced generative adversarial networks (organ) for sequence generation models. arXiv preprint arXiv:1705.10843 (2017)
13. Haddadi, H.: Fighting online click-fraud using bluff ads. ACM SIGCOMM Computer Communication Review **40**(2), 21–25 (2010)
14. Haider, C.M.R., Iqbal, A., Rahman, A.H., Rahman, M.S.: An ensemble learning based approach for impression fraud detection in mobile advertising. Journal of Network and Computer Applications **112**, 126–141 (2018)
15. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Advances in neural information processing systems. pp. 6626–6637 (2017)
16. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144 (2016)
17. Kapoor, K.K., Dwivedi, Y.K., Piercy, N.C.: Pay-per-click advertising: A literature review. The Marketing Review **16**(2), 183–202 (2016)

18. Killoran, N., Lee, L.J., Delong, A., Duvenaud, D., Frey, B.J.: Generating and designing dna with deep generative models. arXiv preprint arXiv:1712.06148 (2017)
19. Kudugunta, S.: Deep neural networks for bot detection. Information Sciences **467**, 312–322 (2018)
20. Kusner, M.J., Hernández-Lobato, J.M.: Gans for sequences of discrete elements with the gumbel-softmax distribution. arXiv preprint arXiv:1611.04051 (2016)
21. Mouawi, R., Elhajj, I.H., Chehab, A., Kayssi, A.: Crowdsourcing for click fraud detection. EURASIP Journal on Information Security **2019**(1), 11 (2019)
22. Nagaraja, S., Shah, R.: Clicktok: click fraud detection using traffic analysis. In: Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks. pp. 105–116 (2019)
23. Neunert, M., Abdolmaleki, A., Wulfmeier, M., Lampe, T., Springenberg, J.T., Hafner, R., Romano, F., Buchli, J., Heess, N., Riedmiller, M.: Continuous-discrete reinforcement learning for hybrid control in robotics. arXiv preprint arXiv:2001.00449 (2020)
24. Oentaryo, R., Lim, E.P., Finegold, M., Lo, D., Zhu, F., Phua, C., Cheu, E.Y., Yap, G.E., Sim, K., Nguyen, M.N., et al.: Detecting click fraud in online advertising: a data mining approach. The Journal of Machine Learning Research **15**(1), 99–140 (2014)
25. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. Advances in neural information processing systems **29**, 2234–2242 (2016)
26. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
27. Thomas, K., Crespo, J.A.E., Rasti, R., Picod, J.M., Phillips, C., Decoste, M.A., Sharp, C., Tirelo, F., Tofigh, A., Courteau, M.A., et al.: Investigating commercial pay-per-install and the distribution of unwanted software. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 721–739 (2016)
28. Yu, L., Zhang, W., Wang, J., Yu, Y.: Seqgan: Sequence generative adversarial nets with policy gradient. In: Thirty-first AAAI conference on artificial intelligence (2017)
29. Zhao, P., Shui, T., Zhang, Y., Xiao, K., Bian, K.: Adversarial oracular seq2seq learning for sequential recommendation. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI. pp. 1905–1911 (2020)
30. Zheng, P., Yuan, S., Wu, X., Li, J., Lu, A.: One-class adversarial nets for fraud detection. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 1286–1293 (2019)