國立政治大學資訊管理學系

碩士學位論文

指導教授: 郁方 博士

HiSeqGAN: 高維資料的序列合成與預測

HiSeqGAN: High-dimensional Sequence Synthesis and Prediction

研究生：田韻杰

中華民國 一〇八年 六月

DOI:10.6814/NCCU201900564

Abstract

High-dimensional data sequences constantly appear in practice. State-of-the-art models such as recurrent neural networks suffer prediction accuracy from complex relations among values of attributes. Adopting unsupervised clustering that clusters data based on their attribute value similarity results data in lower dimensions that can be structured in a hierarchical relation. It is essential to consider these data relations to improve the performance of training models. In this work, we propose a new approach to synthesize and predict sequences of data that are structured in a hierarchy. Specifically, we adopt a new hierarchical data encoding and seamlessly modify loss functions of SeqGAN as our training model to synthesize data sequences. In practice, we first use the hierarchical clustering algorithm, GHSOM, to cluster our training data. By relabelling a sample with the cluster that it falls to, we are able to use the GHSOM map to identify the hierarchical relation of samples. We then converse the clusters to the coordinate vectors with our hierarchical data encoding algorithm and replace the loss function with maximizing cosine similarity in the SeqGAN model to synthesize cluster sequences. Using the synthesized sequences, we are able to achieve better performance on high-dimension data training and prediction compared to the state-of-the-art models.

Keywords: High-dimensional data, Sequence Synthesis, Sequence Prediction, SeqGAN, GHSOM.

DOI:10.6814/NCCU201900564

Contents

DOI:10.6814/NCCU201900564

List of Figures

DOI:10.6814/NCCU201900564

List of Tables

DOI:10.6814/NCCU201900564

1 Introduction

With the progress and the fast development of technology, artificial intelligence forecasting techniques have been receiving much attention in recent years. Deep learning is one of advanced machine learning technique which is highly used in artificial intelligence. It relies on sophisticated mathematical and statistical computations for solving complicated real-world problems. As we know, AlphaGo [1], an application of deep neural network have gained victory with a Chinese Go master lately. Moreover, there have been numerous applications of deep learning in our lives such as automatic classification of emails [2] [3], chatbot [4] and image recognition [5]. In different applications, we can also use different models to train machines. For example, convolutional neural network (CNN) [6], recurrent neural network (RNN) [7] and generative adversarial network (GAN) [8] are some of the modern implementations of deep learning.

CNN is a model which is most commonly applied to analyze visual imagery. It is built by the visual cortex of the human brain and consists of convolutional layers and pooling layer. The former applies a convolution operation to the input and pass the result to the next layer. And then the convolution emulates the response of an individual neuron to visual stimuli. The latter combines the outputs of neuron clusters at one layer into a single neuron in the next layer. Through the structure, CNN can optimize the correlation between pattern recognition and adjacent data, which shows excellent performance on recognition especially in the data type of image [5] and sound [9].

Sequence modeling has been one of the most complicated tasks in real-world problems. When it comes to sequence modeling, noise, the length of time and pattern variabilities always hamper the progress. RNN is a model that heavily used in sequence modeling. It is different from the general feedforward neural network since the neurons in the RNN have a temporary internal memory that remembers the previous output state, and then the neuron can calculate different output values based on the previous state. Also because RNN can remember the feature of a previous output state, this model can handle input data of different lengths, which has a good performance in applications such as automatic

DOI:10.6814/NCCU201900564

translation [10] and speech recognition [11].

However, RNN encounters vanishing gradient problem and exploding gradient problem in the practice training. These problems lead the stochastic gradient descent method to produce volatile results, and this situation also affects the performance of analyzing longer sequence data. In order to solve these problem encountered by RNN, long-short-term memory (LSTM) is proposed by Hochreiter et al. [12]. In the neurons of LSTM, three gates for input, forget, and output is added to this model. After inputting the data, it determines the opening and closing of each gate according to the respective weight calculation results. According to Gers et al. [13], when training data are longer sequences, the forget gate can reset the neuron state to zero or reduce the value of the neuron state slowly, thereby effectively avoiding the problem of over-amplification of the neuron.

GAN is proposed by Goodfellow et al. in 2014[8] and has shown excellent performance for data generation. It is an unsupervised learning method in deep learning that learns through two neural networks, generator and discriminator. The generator samples the data randomly as input from the latent pace, and it is just like a forger who tries to mimic the real-world data in training while the discriminator is just like a policeman who tries to distinguish the fake data generated by the generator. Through iteratively training on generator and discriminator, both the generator and the discriminator advance. In other words, the goal of the generator is to maximize the probability of discriminator making a mistake. We expect that the discriminator cannot judge whether the output of the generator is correct.

Nevertheless, GAN has limitations on training discrete sequences. The first reason is that the generator usually needs an LSTM model, so when the generator passes data to the discriminator, it gets a sequence of discrete values, which makes the gradient update challenging to handle. The second reason is that the discriminator can only accept a complete sentence. The generation process is a sequential decision process, and it is crucial to balance its current score and future score. The sequence generative adversarial nets

DOI:10.6814/NCCU201900564

(SeqGAN) [14] adopts reinforcement learning and policy gradient to solve this problem. It is the first work that extends GAN to generate sequences of discrete tokens.

However, for high-dimensional data sequences, we have observed that SeqGAN suffers from complex computations on relations of attributes. It is challenging to model and draw data samples from high dimensional distributions. We can learn the parameters of conditional probability distributions that map intermediate, latent variables from simpler distributions to more complex ones [15]. Some researches also use the learned intermediate representations on retrieval and classification [16] [17] [18] [19]. Myszkowski et al. [20] use hierarchical clustering to build a hierarchy of documents which gives them a useful advantage in navigation of document search on visual content.

In this work, we propose a two phase analysis. We first adopt growing hierarchical self-organizing maps (GHSOM) as the unsupervised clustering means to construct high-dimensional data into hierarchical clusters. Samples that fall into the same cluster have relatively similar attribute values. This enables us to use clusters, instead of the high-dimension values, to represent the input data, and turn the problem of synthesizing high-dimensional data sequences into the problem of synthesizing hierarchical data sequences. In the second phase of our analysis, we propose HiSeqGAN that adopts the SeqGAN model to synthesize sequences that have their data in a hierarchical relation. To this aim, we propose a novel coordinate encoding to represent clusters in a hierarchical relation. We modify the reward function in HiSeqGAN on maximizing sequence cosine similarity to integrate the hierarchical relations of clusters into SeqGAN. Compared to loss function on mean square errors, considering the hierarchical relations improves the quality of data that we have generated. In our experiments, we showed that the synthesized sequences can be used as new input data to train a better RNN model. Furthermore, we synthesize sequences that have lengths longer than the training set, and show how to use them to predict the future movement of sequences that are similar to the prefix of synthesized sequences. Finally, we discuss how to predict actual attribute values from the predicted cluster. We are able to achieve better accuracy compared to the state-of-the-art model.

DOI:10.6814/NCCU201900564

2 Related Work

Many researchers have shown interest in the generative aspects of CNN recently to figure out what the model learn and how to improve the model. Furthermore, the generative models can be beneficial when the training data are not enough, and the labeled data are incomplete. Dai et al. [21] have constructed a generative model for the CNN and have proposed a method of visualization which can directly draw synthetic samples for any given node in a trained CNN by the Hamiltonian Monte Carlo (HMC) algorithm. Xie et al. [22] used the Gaussian distribution which is used initially in convolutional neural networks as an example to simulate the way of probability distribution in the process of CNN. Moreover, they used auto-encoder which is commonly used in unsupervised learning and Langevin dynamics algorithm to learn reconstructed pictures. Also, they found a generative random field model has the potential to learn from big unlabeled data. After that, they proposed a CoopNets [23] which can train a bottom-up descriptor network and a top-down generator network simultaneously. Both the descriptor and the generator are involved in Langevin sampling and are in the form of alternating back-propagation.

Recently, GAN has shown excellent performance especially in the field of computer visions. However, Dai et al. [24] explained the doubts about semi-supervised learning with GAN. In their research, they improved the drawbacks of feature matching GAN and presented a semi-supervised learning framework. Also, Salimans et al. [18] focused on semi-supervised learning and the generation of images in their research. Moreover, they proposed an evaluation metric for comparing the quality of these models. Radford et al. [17] proposed a deep convolutional generative adversarial networks (DCGAN) and demonstrated its efficacy through image data, which proved that unsupervised learning has excellent and stable results. Besides, it is an essential issue for models to produce high-resolution images. Denton et al. [25] proposed the LAPGAN model with a Laplacian pyramid framework to create a pyramid structure. The model uses a generation network at each level of the pyramid to produce a higher resolution image.

In addition to improving and optimizing the GAN model, many research proposed a

DOI:10.6814/NCCU201900564

variant based on the original architecture. Ho et al. [26] have implemented the imitation learning of deep reinforcement learning in a given environment. They combined this method with GAN, and it can ultimately be implemented in robots and self-driving vehicles. Luc et al. [27] applied the GAN model to semantic segmentation for the first time. They add a discriminator to the semantic segmentation model and finally proved that this method could improve the consistency of high-order potentials in their experiments. Liu et al. [28] proposed coupled generative adversarial networks (CoGAN)which enforced a weight sharing constraint on learning a joint distribution of multi-domain images. Finally, they applied this model to several learning tasks, including color and depth images, and face images with different attributes. All of the tasks show successful results by learning the joint distribution without any tuple of corresponding images. Besides, Zhu et al. [29] proposed a cycle-consistent adversarial network (CycleGAN) in conjunction with the concept of cycle consistency. They presented a method that can learn to capture the unique characteristics of one images collection and to translate these characteristics into the other image collection.

Besides, Tulyakov et al.[30], Saito Matsumoto et al.[31] and Vondrick et al.[32] have researched for recent video generative models based on GAN. Nevertheless, GAN does not infer the latent noise vectors while VAE needs to design an inference model for the sequence of noise vectors, which is a non-trivial task due to the complex dependency structure. Xie et.al[33] proposed a learning dynamic generator model, using alternating back-propagation through time to learn realistic models for dynamic textures and action patterns. It does not require an extra model such as a discriminator in GAN or an inference model in VAE.

DOI:10.6814/NCCU201900564

3 Hierarchical Data

The first phase of our analysis is to cluster high-dimensional data into a hierarchical relation. In this way, we are able to reduce the data dimension. Similar to symbolic dynamics, we seek for a partition of the state space that describes the trajectories of points to represent the dynamics in a simple way. However, it is hard to extract the best partition associated to Markov process [34]. In this paper, we propose hierarchical clustering to achieve unsupervised partition, where number of partitions is dynamically determined based on sample variations. Particularly, we adopt growing hierarchical self-organizing map (GHSOM) algorithm [35] to cluster our data first, and then investigate hierarchical data sequence model training in the second phase. One can predict actual attribute values from samples that fall into the predicted cluster as we show in the experiments.

3.1 Growing Hierarchical Self-Organizing Map(GHSOM)

Unsupervised clustering is one common way to reduce data dimensions. GHSOM [35] is a self-organizing map that grows hierarchically based on data distribution. It expands self-organization maps in a hierarchy according to the variance between and within clusters. When the horizontal and vertical expansion thresholds are set, the algorithm continuously grows the map and checks whether variations between the clusters and within the clusters meet the set requirement. Unlike K-means [36] or SOM [37] where number of clusters is given in advance, using GHSOMs, given tolerances on variations of between and within clusters, samples are separated and clustered iteratively on the fly until the given tolerances are satisfied.

The constructions for using GHSOM consists of the following four steps: 1) Initialize layer 0: Layer 0 includes a single node, the weight vector of which is initialized as the expected value of all input data. The mean quantization error of layer 0(MQE_0) is calculated next. The MQE of a node denotes the mean quantization error that sums the deviation between the weight vector of the node and all input data mapped to the node. 2) Train each individual SOM: As a component of the training process of an individual SOM,

DOI:10.6814/NCCU201900564

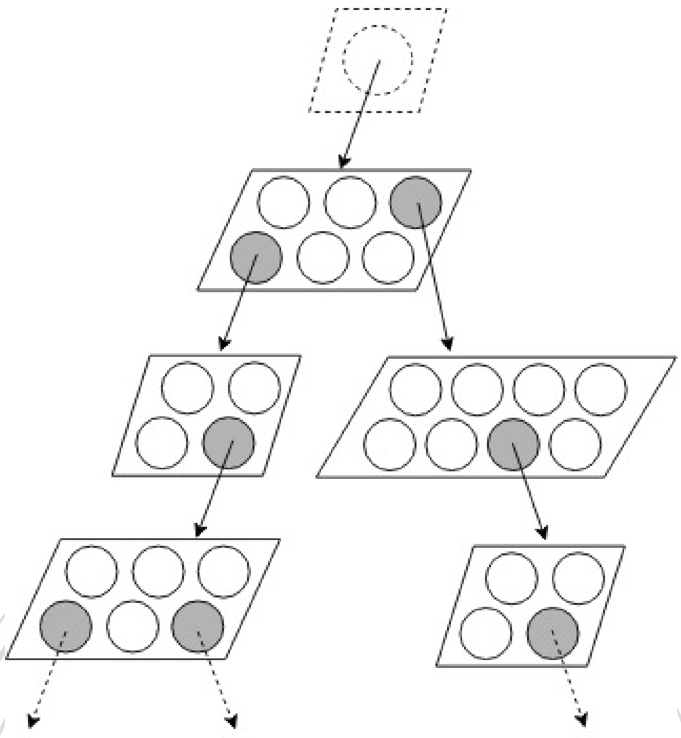
Figure 1: GHSOM structure

the input data are imported individually. The distances between the imported input data and the weight vectors of all nodes are calculated, and the node with the shortest distance is selected as the winner. Under the competitive learning principle, only the winner and its neighboring nodes qualify for adjustment of their weight vectors. The competition and training processes are repeated until the learning rate decreases to a certain value. 3) Grow each individual SOM horizontally: Each individual SOM grows until the mean value of the MQEs for all nodes on the SOM is smaller than the MQE of the parent node multiplied by parameter τ_1. If the stop criterion is not satisfied, we find the error node that owns the largest MQE and insert one row or column of new nodes between the error node and its dissimilar neighbor. 4) Expand or terminate the hierarchical structure: The node with its MQE greater than $\tau_2 \times MQE_0$ will be used to develop the next SOM layer.

DOI:10.6814/NCCU201900564

Algorithm 1 GHSOM Algorithm

```
Initialize the parameters $\tau_1$ and $\tau_2$.
Initialize the set of layers $L = \{l_0\}$.
Initialize the set of maps $M = \{m_0\}$.
Randomly initialize the weight vectors $w_i$ and normalize the values.
Compute the initial quantization error $qe_0$
for $l$ in $L$ do
  for $m$ in $M$ do
    repeat
      Train the map $m$ as a single SOM
      Compute $qe_i$ and $MQE_m$
      if $MQE_m \geq \tau_1 \cdot qe_u$ then
        Select a neuron $e$ with the highest $qe$ and its most dissimilar neighbor $d$.
        Insert a row or column of neurons between $d$ and $e$ and initialize their weight
        vectors as the means of their respective neighbors.
      end if
    until $MQE_m < \tau_1 \cdot qe_u$
    for $w_i$ in $W$ do
      if $qe_i \geq \tau_2 \cdot qe_0$ then
        Create a new map in the lower layer from the neuron $i$.
      end if
    end for
  end for
end for
```

DOI:10.6814/NCCU201900564

We use the GHSOM toolbox [38] to implement the clustering process with GHSOM. Segment of the algorithm is shown in Algorithm 1. By adjusting the values of breadth(τ_1) and depth(τ_2), one may derive different shapes of GHSOMs and discover the suitable one. To evaluate the partitioning of GHSOM, we can calculate the entropy rate according to [39][34]. Table 1 shows the entropy rate comparison with different clustering method.

Table 1: Entropy rate of different clustering methods

Method	K-means	SOM	GHSOM
entropy rate	4.191	3.926	3.048

3.2 Hierarchical Data Encoding

The GHSOM map results data in clusters of a tree-like structure. We use decimal encoding [40] to label the clusters. Each digit in the decimal number corresponds to a cluster of a layer. We append zero at the end of the decimal number as a padding symbol to indicate clusters that have no child clusters in the next layer. As shown in Fig. 2, labels 230, 164 and 475 represent some clusters in a three layer map, and the 230 refers to a cluster that does not has any of its child clusters in the third layer.

After using the decimal number to label clusters, we converse each number to the two-dimensional coordinate vector. The idea is to squeeze all the clusters in one square layer by layer as shown in Fig. 3. Each label can then be encoded as the coordinate of the center point of each square that represents to the cluster as shown in Table 2.

DOI:10.6814/NCCU201900564

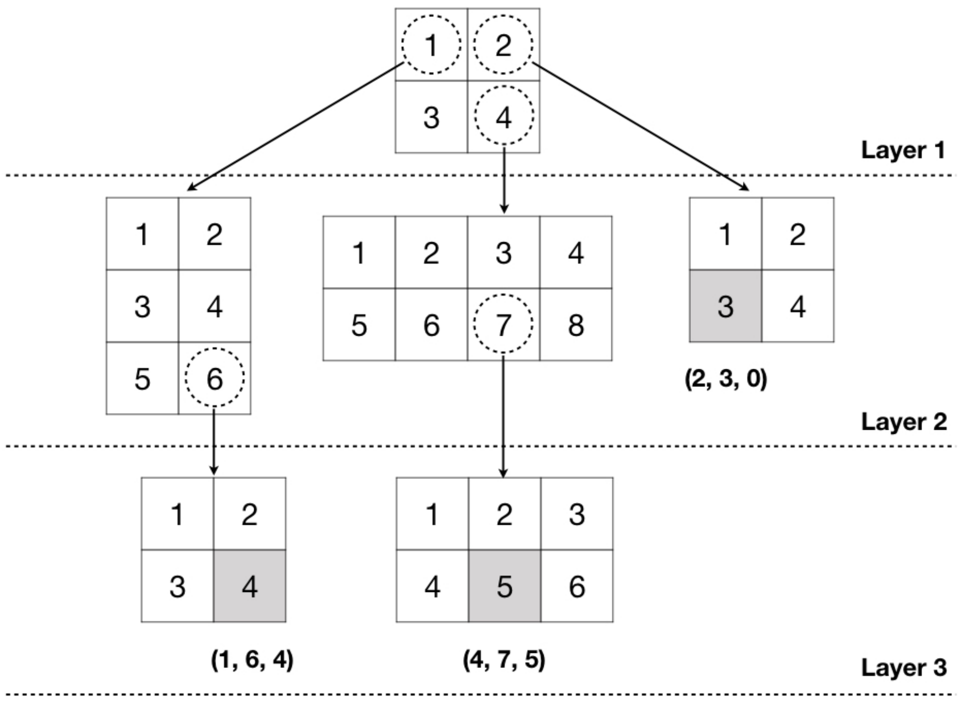

Figure 2: Decimal number labelling on clusters of a GHSOM map

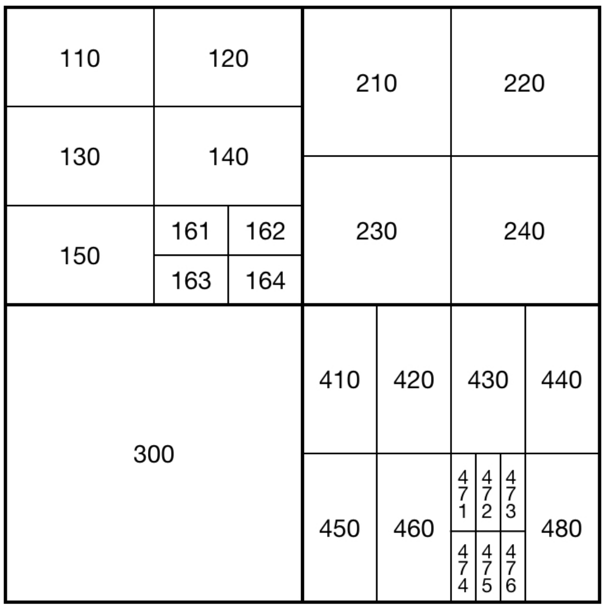

Figure 3: Coordinate encoding of clusters with decimal labels

DOI:10.6814/NCCU201900564

Table 2: Coordinate points of GHSOM clusters

Cluster	Point	Cluster	Point	Cluster	Point
110	$(\frac{1}{8}, \frac{11}{12})$	210	$(\frac{5}{8}, \frac{7}{8})$	450	$(\frac{9}{16}, \frac{1}{8})$
120	$(\frac{3}{8}, \frac{11}{12})$	220	$(\frac{7}{8}, \frac{7}{8})$	460	$(\frac{11}{16}, \frac{1}{8})$
130	$(\frac{1}{8}, \frac{3}{4})$	230	$(\frac{5}{8}, \frac{5}{8})$	471	$(\frac{37}{48}, \frac{3}{16})$
140	$(\frac{3}{8}, \frac{3}{4})$	240	$(\frac{7}{8}, \frac{5}{8})$	472	$(\frac{13}{16}, \frac{3}{16})$
150	$(\frac{1}{8}, \frac{7}{12})$	300	$(\frac{1}{4}, \frac{1}{4})$	473	$(\frac{41}{48}, \frac{3}{16})$
161	$(\frac{5}{16}, \frac{5}{8})$	410	$(\frac{9}{16}, \frac{3}{8})$	474	$(\frac{37}{48}, \frac{1}{16})$
162	$(\frac{7}{16}, \frac{5}{8})$	420	$(\frac{11}{16}, \frac{3}{8})$	475	$(\frac{13}{16}, \frac{1}{16})$
163	$(\frac{5}{16}, \frac{13}{24})$	430	$(\frac{13}{16}, \frac{3}{8})$	476	$(\frac{41}{48}, \frac{1}{16})$
164	$(\frac{7}{16}, \frac{13}{24})$	440	$(\frac{15}{16}, \frac{3}{8})$	480	$(\frac{15}{16}, \frac{1}{8})$

Taking cluster 164 as an example, the first layer has four clusters, and the first layer of 164 is 1, i.e., the first cluster. Thus, the two-dimensional coordinate we assign to its interval vector is: $(\frac{1}{4}, \frac{3}{4})$. The second layer has a total of six clusters, and the second layer of 164 is 6, i.e., the sixth cluster. The two-dimensional coordinate of the layer vector is: $(\frac{3}{8}, \frac{7}{12})$. The third layer has four clusters, and the the third layer of 164 is 4, i.e., the fourth cluster. The vector is: $(\frac{5}{16}, \frac{5}{8})$. which is same with the cluster 164 by using decimal number encoding.

Eq. (1) to Eq. (4) show how we calculate coordinates of the center point of each square that represents a cluster in a GHSOM map. P_x (P_y) is the coordinate of x-axis (y-axis) that we aim to calculate for the center point of the square that represents the label of the target cluster; m refers to the number of the GHSOM layers; P_i^x (P_i^y) refers to the coordinate in thei_{th} layer and B_i^x (B_i^y) refers to the unit width of x-axis (y-axis) of the square in the i_{th} layer, where x_i (y_i) refers to the position of the target cluster in the i_{th} layer. Both B_0^x and B_0^y are set to 1, i.e., the square of the whole space is initially set to 1, where its central point is set (0.5, 0.5). n_i^x (n_i^y) refers to size of the SOM map, i.e., the number of x-axis (y-axis) clusters, in the i_{th} layer; r_x (r_y) refers to the offset as the width of x-axis (y-axis) of the last square (the last layer of the cluster). The coordinate of x-axis P_x (of y-axis P_y), can be calculated using the following formulas.

DOI:10.6814/NCCU201900564

$$P_x = r_x + \sum_{i=1}^{m} P_i^x; \quad P_y = r_y + \sum_{i=1}^{m} P_i^y \tag{1}$$

, where

$$B_i^x = B_{i-1}^x \times \frac{1}{n_i^x}; \quad B_i^y = B_{i-1}^y \times \frac{1}{n_i^y} \quad (B_0^x = 1 \quad and \quad B_0^y = 1) \tag{2}$$

$$P_i^x = B_i^x \times x_i; \quad P_i^y = B_i^y \times y_i \tag{3}$$

$$r_x = B_m^x \times \frac{1}{2}; \quad r_y = B_m^y \times \frac{1}{2} \tag{4}$$

Algorithm 2 shows more details how we converse the clusters to the two-dimensional coordinates.

Algorithm 2 Transformation of 2-dimensional coordinate Algorithm

```
Input :
    1. Amount of layers $m$
    2. Number of GHSOM layers $i$ $\{i = 1, ...., m\}$
    3. Size of x-axis, y-axis in $i$ layer $n_{xi}$, $n_{yi}$
    4. The x coordinate, y coordinate in $i$ layer $x_i$, $y_i$
Output :
    1. The x coordinate in GHSOM coordinates $P_x$
    2. The y coordinate in GHSOM coordinates $P_y$
1: Initialize $B_0^x, B_0^y \leftarrow 1$
2: Use Eq. (2) to Calculate the equal segment of x and y
3: Set $P_i^x$ and $P_i^y$ by Eq. (3)
4: Get $P_x$ and $P_y$ by Eq. (1)
```

DOI:10.6814/NCCU201900564

4 Data Synthesis and Prediction with HiSeqGAN

In the second phase of our analysis, we propose HiSeqGAN as our primary training model on hierarchical data sequence synthesis. HiSeqGAN adopts the sequence generation framework of SeqGAN [14] but with its optimization function on maximizing sequence similarity based on cosine similarity of coordinates to take the hierarchical relations among data into account.

4.1 HiSeqGAN

Fig. 4 shows the structure of HiSeqGAN, similar to the proposed framework in [14]. The generator used in the sequence generation is an RNN model with LSTM cells, while the discriminator is a CNN model. First, HiSeqGAN uses the real-world data and the native samples generated by the generator (fake samples) to train the discriminator, so that the discriminator can distinguish real-world data from fake samples. Then, the reward is passed back to the intermediate state-action steps by using the Monte Carlo search. Second, the generator updates by reinforcement learning (RL). To be more specific, the generator is treated as an RL agent; the state refers to the currently produced tokens; the action refers to the next token to be generated. In a stochastic parameterized policy, the actions are drawn from a distribution that parameterizes the policy and the objective of the policy is to generate a sequence from the start state in such a way that maximizes the expected end reward.

DOI:10.6814/NCCU201900564

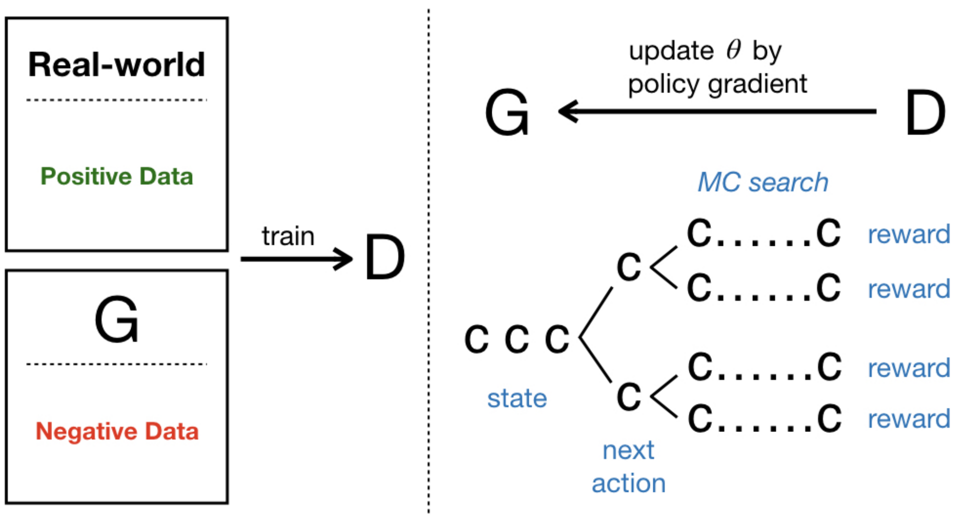

Figure 4: The HiSeqGAN framework

4.2 Sequence Similarity

As defined in section 3, we represent the cluster in a GHSOM map as the coordinates of the center point of its square. To evaluate the prediction on clusters between two labels, we propose three-dimensional cosine similarity on their coordinates. The reason that we extend two-dimensional to three-dimensional is to prevent the distortion of the original point on the same plane. As shown in Fig. 5 and Fig. 6, the 2D cosine similarity of point C_1 and point C_2 is larger than point C_1 and point C_3 at two-dimensional coordinate while the variance between point C_1 and point C_3 is smaller since they have the same first layer in the hierarchical cluster. To increase the accuracy of our generated points, we add a dimension to separate the points and the origin of the coordinate at the different planes. That is to say, point C_1, point C_2 and point C_3 are transformed into $(\frac{3}{8}, \frac{3}{4}, 1)$, $(\frac{5}{8}, \frac{7}{8}, 1)$ and $(\frac{7}{16}, \frac{13}{24}, 1)$ respectively while O is $(0, 0, 0)$. Then, the cosine similarity of point C_1 and point C_3 is larger than point C_1 and point C_2 at a three-dimensional coordinate, which means the variance between point C_1 and point C_3 is also smaller.

DOI:10.6814/NCCU201900564

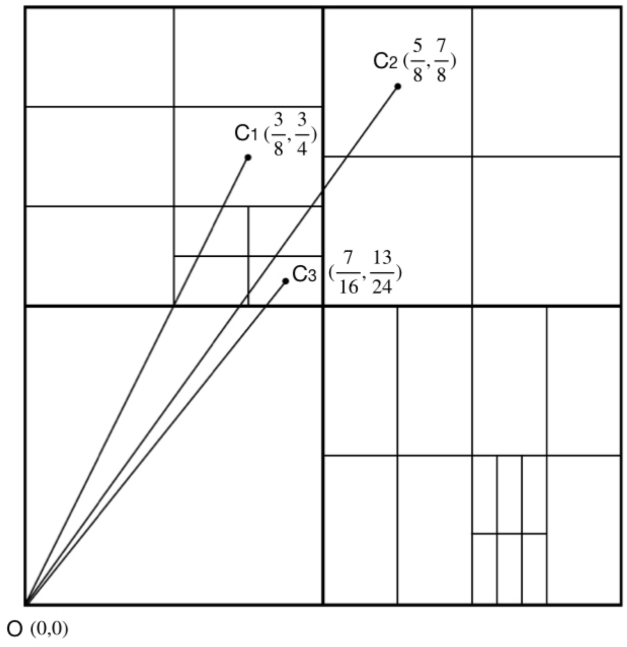
C2 $(\frac{5}{8},\frac{7}{8})$
C1 $(\frac{3}{8},\frac{3}{4})$
C3 $(\frac{7}{16},\frac{13}{24})$
O (0,0)

Figure 5: Clusters with 2D coordinates

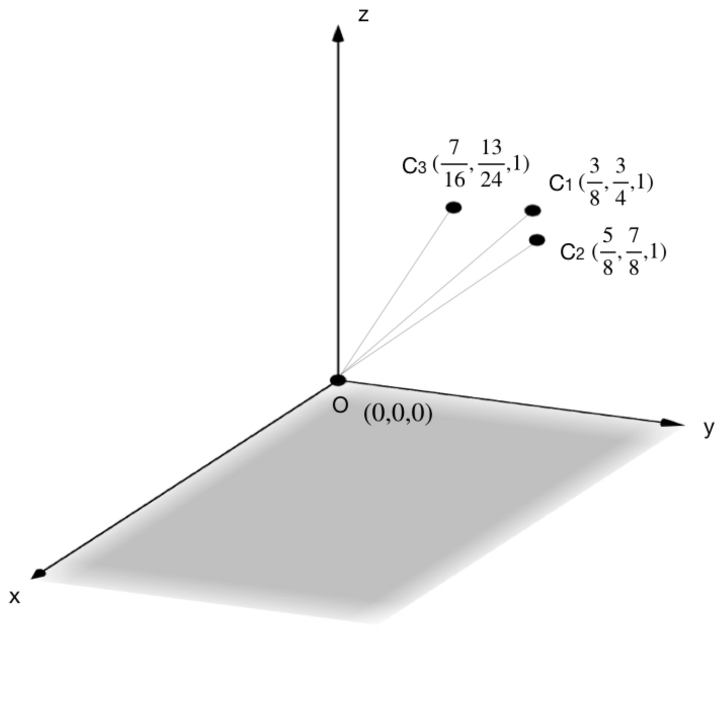
z
C3 $(\frac{7}{16},\frac{13}{24},1)$
C1 $(\frac{3}{8},\frac{3}{4},1)$
C2 $(\frac{5}{8},\frac{7}{8},1)$
O (0,0,0)
y
x

Figure 6: Clusters with 3D coordinates

DOI:10.6814/NCCU201900564

Given the synthesized sequence as $Y_{1:t} = (y_1, ..., y_t)$ and $Y^c_{1:t} = (y^c_1, ..., y^c_t)$ as raw sequence of clusters. We define the metric of sequence similarity as Eq. (5), where $\vec{p}_i$ and $\vec{p}^{\,c}_i$ are the three-dimensional vectors, which refer to the coordinate transformation of y_i and y^c_i respectively.

$$\texttt{SequenceSimilarity}(Y_{1:t}, Y^c_{1:t}) = \sum_{i=1}^{t} \frac{\vec{p}_i \cdot \vec{p}^{\,c}_i}{\left\|\vec{p}_i\right\| \left\|\vec{p}^{\,c}_i\right\|} \tag{5}$$

4.3 Sequence Synthesis

Following SeqGAN, our generator model is updated by employing a policy gradient and Monte Carlo search, where the final reward signal is provided by the discriminator based on Eq.(5) and is passed back to the intermediate action value. Given a dataset, train a generative model G_θ of parameter θ to produce a sequence $Y_{1:T} = (y_1, ..., y_t)$, and train a discriminative model D_ϕ of the parameter ϕ to distinguish real or fake data for improving the generator. Then, we use an N time Monte Carlo search with a roll-out policy G_β to sample the unknown last $T-t$ tokens. Eq.(6) shows how we use the roll-out policy to get a batch of output samples.

$$Q^{D_\phi}_{G_\theta}(s = Y_{1:t-1}, a = y_t) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y^n_{1:T}), (Y^n_{1:T}) \in MC^{G_\beta}(Y_{1:t}; N), & \textit{for } t < T \\ D_\phi(Y_{1:t}), & \textit{for } t = T \end{cases} \tag{6}$$

To keep a good pace with the generator, the discriminator shall re-train as long as the generator generates more realistic sequences.

$$\min_\phi -\mathbb{E}_{Y\sim_p}[\log D_\phi(Y)] - \mathbb{E}_{Y\sim G_\theta}[\log(1 - D_\phi(Y))] \tag{7}$$

DOI:10.6814/NCCU201900564

After a new discriminator model has been obtained, we update the generator for optimizing a parameterized policy to maximize the long-term reward. Following , the objective function can be derived as Eq.(8)

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{Y_{1:t-1 \sim G_\theta}} \left[\sum_{y_t \epsilon y} \nabla_\theta G_\theta(y_t | Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t) \right] \tag{8}$$

Then, the parameters of generator can be updated by Eq.(9). α_h refers to the corresponding learning rate at i^{th} step.

$$\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta) \tag{9}$$

Algorithm 3 shows full details of our model. Firstly, we use the maximum likelihood estimation (MLE) to pre-train generator on the training dataset. Secondly, we pre-train discriminator by maximizing the cosine similarity, the output of the pre-trained generator can be our negative samples whereas the given dataset is our positive examples. Next, we re-train our discriminator when the generator has improvement. Also, to reduce the variability of the estimation, we combine different datasets of negative samples with positive ones.[41]

DOI:10.6814/NCCU201900564

Algorithm 3 Sequence Synthesis Algorithm

```
 1: Initialize $G_\theta$, $D_\phi$ with random parameter $\theta$, $\phi$
 2: Pre-train $G_\theta$ via MLE
 3: Assign parameter $\theta$ to the roll-out policy
 4: Pre-train $D_\phi$ by Eq. (5)
 5: repeat
 6:     for g-steps do
 7:         Generate a sequence
 8:         for t in 1:T do
 9:             Compute $Q$ by Eq. (6)
10:         end for
11:         Update $\theta$ by Eq. (9)
12:     end for
13:     for d-steps do
14:         Train $D_\phi$ by Eq. (7)
15:     end for
16:     Update the parameter of roll-out policy
17: until model converges
```

DOI:10.6814/NCCU201900564

4.4 Sequence Prediction

In order to predict next tokens of training sequences, we use the trained generator G_θ to generate sequences that have more periods, and the extended tokens are used as our prediction on sequences that have similar prefix.

Formally speaking, given the length of sequences of our training data, denoted as t, to predict the next t' periods of the sequence, we synthesize sequence $Y = (y_1, ..., y_t, y_{t+1}, ..., y_{t+t'})$.

Given a set of synthesized sequences S, and a real item sequence Y^{item} that has its length t, Eq. (10) defines the way to find a synthesized sequence $Y_{1:t+t'} \in S$ that has its prefix $Y_{1:t}$ most similar to the item sequence Y^{item}. Then, we use the postfix $Y_{t+1:t+t'} = (y_{t+1}, ..., y_{t+t'})$ to predict the future movement.

$$\max_{Y \in S} \texttt{SequenceSimilarity}(Y_{1:t}, Y^{item}) \tag{10}$$

DOI:10.6814/NCCU201900564

5 Experiments

In this section, we evaluate our approach against supply chain management data from real worlds. We use semiconductor component distributors to conduct research. All the experiment data are collected from real transactions of the company W, one of the world's largest semiconductor component distributor in the Asia Pacific area. With more than 30 branches and over 60 international electronics component suppliers, company W plays a key role as a supply chain buffer for a franchise partner, including Intel, Philips, Texas Instruments, Hynix, Vishay and Omni Vision. Our goal is to predict at the item level the values of essential attributes on demand and inventory.

5.1 Data Settings

The raw data of transactions contain more than eighty thousand weekly transaction records over thousands of semiconductor component items and customers. Each item-customer pair has records up to 96 weeks. (Weeks that have no demands have no transaction records). For each record, we label it with nominal attributes including item, customer and date (on week), and use numerical attributes including all the predefined indicators of demand and inventory in supply chain management as the sample attributes of that record. Each item-week record consists of 8 demand and inventory attributes as shown in Table 3.

DOI:10.6814/NCCU201900564

Table 3: Numerical Attributes

Indicators	**Name**	**Description**
Inventory	Actual AWU	Average weekly usage (i.e., actual demand) in the past
	FCST M	Managers forecast of monthly demand for the future
Demand	BL <= 9WKs	In-transit inventory to be delivered by upstream supplier within 9 weeks
	Backlog (BL)	Total in-transit inventory (to be delivered)
	DC-OH	On-hand (OH) inventory in the distribution center (DC)
	Hub-OH	On-hand (OH) inventory in warehouse nearby downstream customer production plant
	TTL OH	In stock quantities
	Available	Company backlog

5.2 From High-dimension Data to Hierarchical Data

We first apply GHSOM to cluster samples based on these numerical attributes. (All the values are normalized with max and min, i.e., n(d) = d-min/max-min.) Fig. 7 shows the partition result, a GHSOM map (in decimal encoding) that consists of three layers with 70 clusters. Based on the GHSOM map, one can label each item-week 8-dimension record with its corresponding cluster. We generate 5712 cluster sequences from raw transaction records, where each cluster sequence refers to an item-customer trajectory movement on demand and inventory attributes. We denote this set as $Data_{raw}$.

DOI:10.6814/NCCU201900564

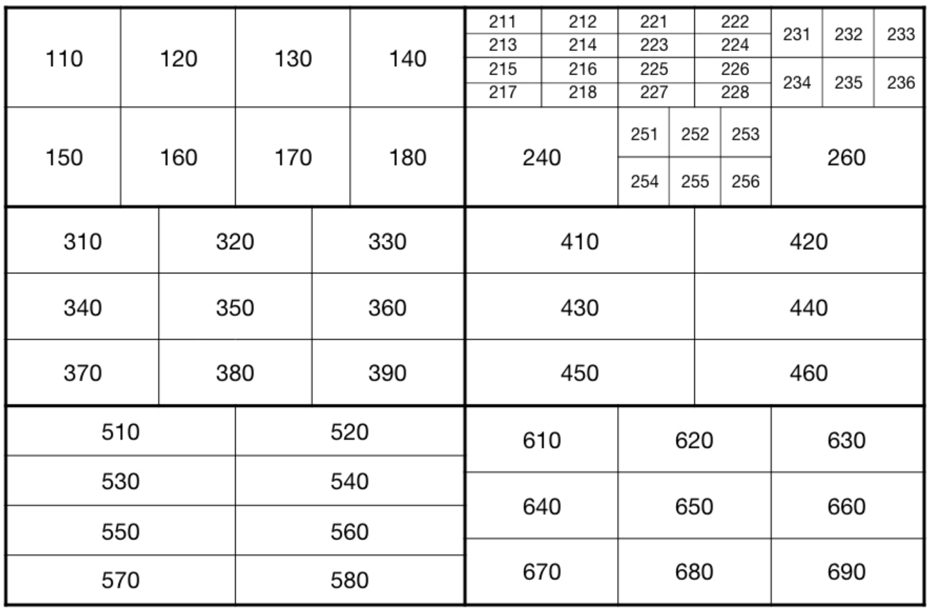

Figure 7: GHSOM clustering result

5.3 Sequence Synthesis

In our first experiment, we use $Data_{raw}$ and the GHSOM map to train our HiSeqGAN model and synthesize another 2500 cluster sequences (denoted as the set $Data_{syn}$). To evaluate the quality of our synthesized sequences, we use RNN as the training model that uses the cluster sequence of week 1 to 95 to predict the cluster in week 96. We first train the RNN model [40] with $Data_{raw}$, the prediction accuracy rate (on average over 5712 item-customer pairs) is 82% on the first layer, 61% on the second layer and 45% on the third layer. That is to say, for an item has its week 96 cluster in 164, the prediction as 1xx (first-layer accurate) is 82%, 16x (second-layer accurate) is 61%, and 164 (third-layer accurate) is 45%. Under the same setting of the RNN, but using both $Data_{raw}$ and $Data_{syn}$ as training inputs, we can improve the precision to 85% on the first layer, 72% on the second layer and 52% on the third layer. The performance is shown in Table 4.

DOI:10.6814/NCCU201900564

Table 4: Layer accuracy of sequence synthesis

Dataset	Layer Accuracy		
	l^1	l^1, l^2	l^1, l^2, l^3
$Data_{raw}$	0.82	0.61	0.45
$Data_{HiSeqGAN}$	0.85	0.72	0.52

The result shows our first contribution that we can synthesize additional training inputs for the state of the art models and improve their performance.

5.4 Sequence Prediction

We conduct experiments on sequence prediction in the second experiment and compare the prediction accuracy against 1) the Naive Bayes method, and 2) the RNN model. The setting is to use week 1-95 to predict week 92-96. The Naive Bayes method [42] is the learning algorithm based on a conditional probabilistic model, which counts observed sequences for prediction. The cluster Naive Bayes learns to predict is the result of creating the probability distribution of all cluster sequence it is shown, and then deciding which cluster to assign on the target week. The RNN has the same setting as the previous experiment but this time it has to use its prediction to predict next periods. For both NaiveBayes and RNN models, we use $Data_{raw}$ with week 1 to 91 for training, and for each item, predict its week 92 cluster, and then with week 2 to 92, to predict week 93, and so on so forth until the prediction on week 96. For our HiSeqGAN model, we use $Data_{raw}$ with week 1 to 91 to synthesize 3840 sequences with week 1 to 96 (denoted as the set S). Then for each item (Y^{item}), we use Eq.(10) to find a synthesized sequence Y that has its prefix best match to Y^{item}, and then use the postfix (clusters on week 92 to 96) of the selected synthesized sequence for prediction.

Table 5 summarizes the results on prediction accuracy on different layers. As one can see that the NaiveBayes model has the worst accuracy rate and drops its accuracy rate

DOI:10.6814/NCCU201900564

significantly after multiple periods. Compared to the HiSeqGAN model, the RNN model has a slight better accuracy rate on the prediction in the first two weeks, but loses its advantage after the third week. This confirms that RNN suffers from error propagation. On the other hand, our HiSeqGAN model is more stable and has better performance on average for long run prediction.

Table 5: Layer accuracy on multi-periods prediction

Method	Accuracy						
	period / layer	92	93	94	95	96	Average
NaiveBayes	l^1	0.28	0.01	0.04	0.06	0.01	0.08
	l^2	0.27	0.0007	0.0017	0.0007	0.0026	0.06
	l^3	0.26	0.0003	0.0001	0	0.0001	0.05
RNN	l^1	0.82	0.71	0.69	0.55	0.53	0.66
	l^2	0.61	0.58	0.51	0.42	0.43	0.51
	l^3	0.46	0.47	0.43	0.42	0.38	0.43
HiSeqGAN	l^1	0.79	0.74	0.74	0.68	0.63	0.72
	l^2	0.62	0.59	0.57	0.54	0.49	0.56
	l^3	0.54	0.51	0.49	0.48	0.43	0.49

5.5 From Cluster Prediction to Actual Value Prediction

Finally, we discuss how we use the predicted cluster to estimate attribute values. This is done by computing the distribution of attribute values with samples that fall into the cluster. For instance, consider that we are interested in the attribute value of AWU (demand on the item). Fig.8 shows the five period prediction on the AWU value of item *itemA*. The points are the actual value of *itemA* in the 92^{nd} to 96^{th} weeks and the

DOI:10.6814/NCCU201900564

boxplots refer to the statistics on AWU values of samples that fall into our predicted clusters. Table 6 shows the mean and standard deviation on AWU of all clusters.

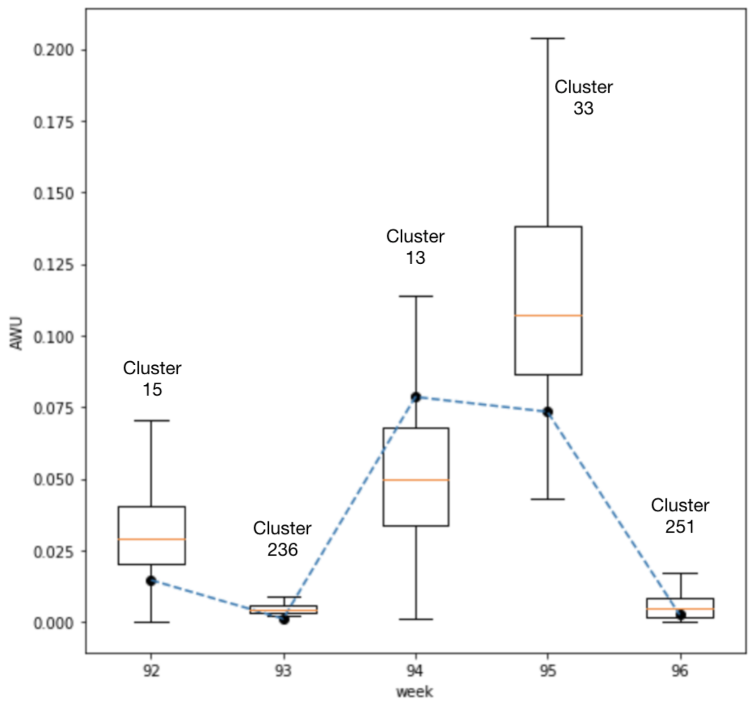

Figure 8: A sampled item for prediction on the AWU value of five periods

Table 6: Mean and standard deviation of AWU

l^1	(μ, σ)	l^1, l^2	(μ, σ)	l^1, l^2, l^3	(μ, σ)
1 (3462)	(0.03774, 0.0284)	11 (245)	(0.09613, 0.02823)	***	
		12 (984)	(0.03397, 0.0135)	***	
		13 (144)	(0.05203, 0.02576)	***	
		14 (124)	(0.0188, 0.01872)	***	
		15 (458)	(0.0308, 0.01476)	***	

DOI:10.6814/NCCU201900564

l^1	(μ, σ)	l^1, l^2	(μ, σ)	l^1, l^2, l^3	(μ, σ)
		16 (789)	(0.01656, 0.01353)	***	
		17 (318)	(0.02714, 0.02031)	***	
		18 (400)	(0.07011, 0.01928)	***	
2 (80693)	(0.00209, 0.00522)	21 (2137)	(0.01128, 0.00824)	211 (293)	(0.00427, 0.00434)
				212 (18)	(0.00249, 0.00182)
				213 (114)	(0.00571, 0.00439)
				214 (317)	(0.00256, 0.00334)
				215 (79)	(0.02048, 0.00495)
				216 (601)	(0.00954, 0.00457)
				217 (392)	(0.02095, 0.00499)
				218 (323)	(0.01788, 0.00463)
		22 (8626)	(0.00478, 0.00354)	221 (1662)	(0.00901, 0.00229)
				222 (1523)	(0.00315, 0.00255)

DOI:10.6814/NCCU201900564

l^1	(μ, σ)	l^1, l^2	(μ, σ)	l^1, l^2, l^3	(μ, σ)
				223 (505)	(0.00598, 0.00369)
				224 (972)	(0.00568, 0.00305)
				225 (290)	(0.00376, 0.0034)
				226 (8)	(0.00338, 0.00457)
				227 (2314)	(0.00274, 0.00192)
				228 (1352)	(0.00404, 0.00343)
		23 (65495)	(0.00052, 0.00111)	231 (49441)	(0.00019, 0.00039)
				232 (1020)	(0.00185, 0.00129)
				233 (1807)	(0.00101, 0.00114)
				234 (5282)	(0.00069, 0.00075)
				235 (5556)	(0.00111, 0.00102)
				236 (2389)	(0.00472, 0.0019)
		24 (1057)	(0.01368, 0.00881)	***	

DOI:10.6814/NCCU201900564

l^1	(μ, σ)	l^1, l^2	(μ, σ)	l^1, l^2, l^3	(μ, σ)
		25 (1681)	(0.00758, 0.0074)	251 (219)	(0.00536, 0.00426)
				252 (194)	(0.00849, 0.00459)
				253 (119)	(0.02612, 0.00731)
				254 (321)	(0.00457, 0.00392)
				255 (641)	(0.00544, 0.00491)
				256 (187)	(0.00994, 0.00662)
		26 (1697)	(0.02496, 0.01107)	***	
3 (1000)	(0.16351, 0.11832)	31 (30)	(0.20976, 0.10459)	***	
		32 (340)	(0.07779, 0.0383)	***	
		33 (98)	(0.11878, 0.04735)	***	
		34 (52)	(0.03807, 0.03842)	***	
		35 (46)	(0.16, 0.0551)	***	
		36 (194)	(0.18049, 0.04735)	***	

DOI:10.6814/NCCU201900564

l^1	(μ, σ)	l^1, l^2	(μ, σ)	l^1, l^2, l^3	(μ, σ)
		37 (69)	(0.29845, 0.08073)	***	
		38 (107)	(0.24879, 0.07085)	***	
		39 (64)	(0.43062, 0.12997)	***	
4 (1598)	(0.03845, 0.0295)	41 (141)	(0.0401, 0.02286)	***	
		42 (168)	(0.06949, 0.03566)	***	
		43 (212)	(0.03875, 0.02146)	***	
		44 (557)	(0.02202, 0.01742)	***	
		45 (148)	(0.07823, 0.03122)	***	
		46 (372)	(0.03242, 0.01885)	***	
5 (338)	(0.36076, 0.17759)	51 (16)	(0.2604, 0.06976)	***	
		52 (16)	(0.30035, 0.07555)	***	
		53 (39)	(0.3325, 0.09421)	***	
		54 (20)	(0.43322, 0.21226)	***	

DOI:10.6814/NCCU201900564

l^1	(μ, σ)	l^1, l^2	(μ, σ)	l^1, l^2, l^3	(μ, σ)
		55 (76)	(0.27628, 0.09706)	***	
		56 (56)	(0.30504, 0.12339)	***	
		57 (72)	(0.59736, 0.13636)	***	
		58 (43)	(0.2382, 0.1169)	***	
6 (347)	(0.13047, 0.09394)	61 (10)	(0.02884, 0.0352)	***	
		62 (9)	(0.17451, 0.0316)	***	
		63 (12)	(0.26544, 0.07343)	***	
		64 (85)	(0.09287, 0.05164)	***	
		65 (5)	(0.54998, 0.0891)	***	
		66 (32)	(0.20099, 0.08159)	***	
		67 (48)	(0.1591, 0.04949)	***	
		68 (79)	(0.06231, 0.03579)	***	
		69 (67)	(0.15814, 0.06932)	***	

DOI:10.6814/NCCU201900564

5.6 Discussion

Note that predicted values have their variation depending on the layer accuracy. The variations are smaller when the accurate layer of the predicted cluster is higher. Since clusters in 1xx contain more samples than clusters in 16x, the second layer accuracy can have better prediction on actual values than the first layer accuracy. We can get most precise estimation when we have the third layer accuracy.

DOI:10.6814/NCCU201900564

6 Conclusion

We present a GAN-like model for synthesizing and predicting sequences of structured data effectively. By using coordinate and cosine similarity to express hierarchical data and loss functions, we are able to adopt SeqGAN to generate more realistic and representative sequences of structured data. In our experiments, we show that the synthesized sequences can be used in two folds: 1) as input data to improve the training process of the state-of-the-art models, and 2) as the prediction of sequences that match the prefix and achieve higher accuracy compared to the state-of-the-art models.

DOI:10.6814/NCCU201900564

References

[1] P. S. Churchland, T. J. Sejnowski, and T. A. Poggio, *The computational brain.* MIT press, 2016.

[2] W. Awad and S. ELseuofi, "Machine learning methods for e-mail classification," *International Journal of Computer Applications*, vol. 16, no. 1, 2011.

[3] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[4] W. B. Rauch-Hindin, *Artificial Intelligence in Business, Science, and Industry: Fundamentals.* Prentice-Hall New Jersey, 1986.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[7] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[9] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

DOI:10.6814/NCCU201900564

[11] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on.* IEEE, 2013, pp. 6645–6649.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] F. A. Gers, J. Schmidhuber, and F. Cummins, *Learning to forget: Continual prediction with LSTM.* IET, 1999.

[14] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient." in *AAAI*, 2017, pp. 2852–2858.

[15] A. Creswell and A. A. Bharath, "Denoising adversarial autoencoders," *IEEE transactions on neural networks and learning systems*, no. 99, pp. 1–17, 2018.

[16] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[17] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[18] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[19] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning.* ACM, 2008, pp. 1096–1103.

[20] P. B. Myszkowski and B. Buczek, "Growing hierarchical self-organizing map for searching documents using visual content," in *2011 Federated Conference on Computer Science and Information Systems (FedCSIS).* IEEE, 2011, pp. 77–81.

DOI:10.6814/NCCU201900564

[21] J. Dai, Y. Lu, and Y.-N. Wu, "Generative modeling of convolutional neural networks," *arXiv preprint arXiv:1412.6296*, 2014.

[22] J. Xie, Y. Lu, S.-C. Zhu, and Y. Wu, "A theory of generative convnet," in *International Conference on Machine Learning*, 2016, pp. 2635–2644.

[23] J. Xie, Y. Lu, R. Gao, S.-C. Zhu, and Y. N. Wu, "Cooperative training of descriptor and generator networks," *arXiv preprint arXiv:1609.09408*, 2016.

[24] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. R. Salakhutdinov, "Good semi-supervised learning that requires a bad gan," in *Advances in Neural Information Processing Systems*, 2017, pp. 6510–6520.

[25] E. L. Denton, S. Chintala, R. Fergus *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in neural information processing systems*, 2015, pp. 1486–1494.

[26] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[27] P. Luc, C. Couprie, S. Chintala, and J. Verbeek, "Semantic segmentation using adversarial networks," *arXiv preprint arXiv:1611.08408*, 2016.

[28] M.-Y. Liu and O. Tuzel, "Coupled generative adversarial networks," in *Advances in neural information processing systems*, 2016, pp. 469–477.

[29] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *arXiv preprint*, 2017.

[30] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz, "Mocogan: Decomposing motion and content for video generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1526–1535.

DOI:10.6814/NCCU201900564

[31] M. Saito, E. Matsumoto, and S. Saito, "Temporal generative adversarial nets with singular value clipping," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2830–2839.

[32] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *Advances In Neural Information Processing Systems*, 2016, pp. 613–621.

[33] J. Xie, R. Gao, Z. Zheng, S.-C. Zhu, and Y. N. Wu, "Learning dynamic generator model by alternating back-propagation through time," *arXiv preprint arXiv:1812.10587*, 2018.

[34] A. Hadriche, N. Jmail, and R. Elleuch, "Different methods of partitioning the phase space of a dynamic system," *International Journal of Computer Applications*, vol. 93, pp. 1–5, 05 2014.

[35] M. Dittenbach, D. Merkl, and A. Rauber, "The growing hierarchical self-organizing map," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 6. IEEE, 2000, pp. 15–19.

[36] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[37] T. Kohonen, *Neurocomputing: Foundations of Research*, J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104428

[38] M. D. E. P. Andreas Rauber, Dieter Merkl, "The growing hierarchical self-organizing map." [Online]. Available: http://www.ifs.tuwien.ac.at/~andi/ghsom/

[39] V. Rajagopalan, A. Ray, R. Samsi, and J. Mayer, "Pattern identification in dynamical systems via symbolic time series analysis," *Pattern Recognition*, vol. 40, no. 11, pp. 2897–2907, 2007.

DOI:10.6814/NCCU201900564

[40] T. Y. Lin, H. H. C. Chuang, and F. Yu, "Tracking supply chain process variability with unsupervised cluster traversal," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech).* IEEE, 2018, pp. 966–973.

[41] J. R. Quinlan *et al.*, "Bagging, boosting, and c4. 5," in *AAAI/IAAI, Vol. 1*, 1996, pp. 725–730.

[42] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan, "Orange: Data mining toolbox in python," *Journal of Machine Learning Research*, vol. 14, pp. 2349–2353, 2013. [Online]. Available: http://jmlr.org/papers/v14/demsar13a.html

DOI:10.6814/NCCU201900564