A GENERATIVE ADVERSARIAL NETWORK FOR GENERATING REALISTIC USERS USING EMBEDDING FROM RECOMMENDATION SYSTEMS

Miss Parichat Chonwiharnphan

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2019

เจเนอเรทีฟแอดเวอเซอเรียลเน็ตเวิร์คสำหรับการสร้างผู้ใช้งานเหมือนจริงโดยใช้การฝังตัวจากระบบแนะนำ

น.ส.ปาริฉัตร ชลวิหารพันธ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2562

Thesis Title	A GENERATIVE ADVERSARIAL NETWORK FOR GENERATING REALISTIC USERS USING EMBEDDING FROM RECOMMENDATION SYSTEMS
By	Miss Parichat Chonwiharnphan
Field of Study	Computer Science
Thesis Advisor	Dr. Ekapol Chuangsuwanich

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Science

ปาริฉัตร ชลวิหารพันธ์ : เจเนอเรทีฟแอดเวอเซอเรียลเน็ตเวิร์คสำหรับการสร้างผู้ใช้งานเหมือนจริงโดยใช้การฝังตัวจากระบบแนะนำ. (A GENERATIVE ADVERSARIAL NETWORK FOR GENERATING REALISTIC USERS USING EMBEDDING FROM RECOMMENDATION SYSTEMS) อ.ที่ปรึกษาหลัก : ดร.เอกพล ช่วงสุวนิช

ข้อมูลลูกค้าหรือผู้ใช้งานเว็บไซต์นั้นถูกใช้ในธุรกิจเพื่อที่จะเข้าใจถึงพฤติกรรมของผู้ใช้งาน และเพื่อหากลยุทธ์ใหม่ที่เป็นประโยชน์ต่อธุรกิจ อย่างไรก็ตามวิธีการทั่วไปนั้นไม่สามารถใช้ได้เมื่อของสิ่งนั้นเป็นสินค้าใหม่ที่ยังไม่มีการตอบรับจากผู้บริโภค เพราะว่าในความเป็นจริงเรายังไม่มีข้อมูลของสินค้าใหม่เหล่านั้น งานวิจัยนี้จัดทำขึ้นเพื่อเสนอวิธีการสร้างผู้ใช้งานที่เหมือนของจริงโดยขึ้นอยู่กับลักษณะของสินค้าใหม่ที่จะวางขาย แบบจำลองของเราใช้เจเนอเรทีฟแอดเวอเซอเรียลเน็ตเวิร์คแบบมีเงื่อนไข (Conditional Generative Adversarial Network: CGAN) และ Straight-Through Gumbel estimator เพื่อให้แบบจำลองของเราสามารถสร้างข้อมูลค่าไม่ต่อเนื่องได้ แบบจำลองของเราจะรับข้อมูลลักษณะของสินค้าใหม่ที่จะวางขาย ซึ่งลักษณะสินค้านั้นจะถูกแปลงให้เป็นเวกเตอร์สินค้าฝังตัว (Product Embedding vector) โดยการใช้ระบบแนะนำ (Recommendation System) ซึ่งเวกเตอร์นี้จะสามารถเก็บความสัมพันธ์ระหว่างสินค้าได้ดีทั้งในมุมของลักษณะสินค้าและมุมมองความชอบของผู้ใช้งาน การทดลองนี้ใช้ข้อมูลการเข้าใช้งานเว็บไซต์อสังหาริมทรัพย์ของผู้ใช้งาน ผลลัพธ์แสดงให้เห็นว่า แบบจำลองของเรามีประสิทธิภาพสูงกว่าอีก 2 วิธี โดยใช้การวัดประสิทธิภาพจาก 4 ตัววัดและยังสามารถสร้างข้อมูลได้เหมือนของจริงแม้จำนวนข้อมูลในบางลักษณะจะต่างกันมากก็ตาม

สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์	ลายมือชื่อนิสิต ....
ปีการศึกษา	2562	ลายมือชื่อ อ.ที่ปรึกษาหลัก ....

6070939521 : MAJOR COMPUTER SCIENCE

KEYWORD: Generative Adversarial Network, Data Generation, Product Embedding, Generative Model, Gumbel-Softmax Distribution

Parichat Chonwiharnphan : A GENERATIVE ADVERSARIAL NETWORK FOR GENERATING REALISTIC USERS USING EMBEDDING FROM RECOMMENDATION SYSTEMS. Advisor: Dr. Ekapol Chuangsuwanich

User data has been used by many companies to understand user behaviors and find new business strategies. However, common techniques could not be used when it comes to new products that have not yet been released due to the fact that there are no prior data available. In this work, we propose a framework for generating realistic user data on new products which can then be analyzed for insights. Our model uses Conditional Generative Adversarial Network (CGAN) with the Straight-Through Gumbel estimator which can also handle discrete-valued outputs. The CGAN is conditioned on product features learned using a recommendation system which can better capture the relationship between products. Experiments using a dataset consisting of view logs from a real estate listing website shows that our model outperforms other baselines on four performance metrics and can effectively predict the finer characteristics of new products.

Field of Study:	Computer Science	Student's Signature ....
Academic Year:	2019	Advisor's Signature ....

ACKNOWLEDGEMENTS

การที่วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดีนั้น นอกจากการทำงานของผู้วิจัยแล้ว ยังมีบุคคลท่านอื่นที่เป็นส่วนสำคัญที่ได้ให้ความช่วยเหลือในการจัดทำวิทยานิพนธ์ฉบับนี้ขึ้นมา ผู้วิจัยรู้สึกซาบซึ้งในความกรุณาเหล่านี้เป็นอย่างมากจึงใคร่ขอใช้เนื้อหาในส่วนกิตติกรรมประกาศของวิทยานิพนธ์ฉบับนี้แสดงความขอบพระคุณเป็นอย่างสูงมา ณ ที่นี้

ขอขอบคุณอาจารย์ที่ปรึกษา ดร.เอกพล ช่วงสุวนิช ผู้ที่คอยให้ความช่วยเหลือและคำปรึกษาให้งานวิจัยและวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วง

ขอขอบคุณ บริษัท homedottech ที่ให้การสนับสนุนข้อมูลเพื่อทำงานวิจัยนี้

สุดท้ายนี้ ผู้วิจัยขอขอบพระคุณบิดามารดา และครอบครัว ซึ่งเปิดโอกาสให้ได้รับการศึกษาเล่าเรียน ตลอดจนคอยช่วยเหลือและให้กำลังใจผู้วิจัยเสมอมาจนสำเร็จการศึกษา

Parichat Chonwiharnphan

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

With the rapid growth of online service, website becomes one of the main channels where people access their favorite content. The data from these websites, often collected in web logs, becomes a source for mining insights about the users, which can be very valuable for business. Techniques such as association mining [1], sequential pattern mining [2], or clustering [3] can be used on web log data to improve profitability. For e-commerce websites, recommendation engines [4] can be trained on the logs to improve conversion rate and thus increase the revenue.

Another business use case for the study of web log is for product development. Based on the web log data, if business can identify the characteristics of target users who are likely to be interested in the new product before its release, they will have more useful insights for product development, user targeting, marketing channel strategy and the best time to launch new campaigns without conducting market survey. The goal of this work is to forecast users' responses to novel products by learning from web log data.

Generative Adversarial Network (GAN) [5] is a generative model for learning from such distribution and generating new realistic samples. It has been successfully applied to computer [6-10] and natural language processing domains [11, 12]. The model consists of two neural networks: a generator and a discriminator. The generator learns to generate new (fake) samples which look similar to the real sample. The discriminator learns to discriminate between real data and fake generated sample and send the feedback to the generator so that it can generate better fake samples.

Moreover, GAN can be trained to control the mode of generated output by adding some kinds of conditional signal, such as class label, to both the generator and the discriminator. This model is called a Conditional Generative Adversarial Network [6]. A relevant work [13] was proposed to apply CGAN for generating novel plausible orders on the e-commerce website in order to forecast the demand, the seasonality, the characteristic of the customer, etc. The input to the model used for conditioning the order is the product name so that the model can be used to estimate orders of new products on the platform. However, the orders that they generated are embeddings that are used as inputs to a separate set of models which will convert the embedding into meaningful information. This is due to the limitation of GAN that it cannot handle discrete value generation well, thus they need to train classifiers to extract the characteristics from generated order representation which makes the training process sophisticated and hard to maintain.

Similarly, our work also tries to generate realistic logs of user for new products, but on the real estate domain. Our data comes from Home.co.th, a real estate listing website in Thailand, which has more than one million views per month. Given a characteristic of a novel real estate, such as the location, the price, the size, the facilities, etc. we generate web logs which can be then aggregated to predict the characteristics of target users for the real estate. Our key technical contributions are enumerated below:

- Recommendation based product embedding: prior works used product descriptions for the conditioning signal. The product description was converted into word embeddings which mean that it can only capture the characteristics of the products as it is explicitly written. It cannot capture other associations such as a person that likes condominiums that are closer to the public transportation system might prefer a condominium with a gym. In order to capture these kinds of relationship, we propose the use of recommender-based embedding which is learned via a deep recommender system.
- A generation approach for web log data with discrete outputs: prior works used a two-step process in order to summarize user characteristics. On the other hand, we handle the limitation of discrete outputs for GAN by using Straight-Through Gumbel Estimator [14, 15] which help simplify the pipeline and reduce error propagation of the models.

For evaluation, it is difficult to measure the quality of generated outputs generated by GAN, especially in non-visual domains. We use four metrics to evaluate the results: Relative Similarity Measure (RSM) [13], Correlation Coefficient (CORR), Earth-Mover Distance (EMD) and Root Mean Squared Error (RMSE). We compare our model with two baseline generation approaches: the nearest neighborhood and a Conditional Variational Autoencoder (CVAE) [16]. The results show that our generator generates the realistic users that are indistinguishable and outperforms the baseline approaches in these metrics.

1.2 Objective

This work aims to build a conditional generative adversarial network for generating realistic users who are likely to visit each project in a website in order to know the characteristic of target users before launch new projects. The performance of this model is better than baseline models.

1.3 Scope of Work

1. This work is trained on web log data from a real estate search engine website to generate users who are likely to visit each project in website.
2. This work generates new users and summarize to the proportion of each characteristic. We did not cover number of visited user estimation or demand forecasting for each project.

1.4 Outcomes

1 An efficiency generation model that takes the characteristic of novel product and outputs the realistic users who are likely to be interested in that product. The generated characteristics of user can be both continuous and discrete value. This result can be summarized to the characteristics of targeted users.

2. To improve the performance of generating realistic users based on particular product, we build the product embedding model by using recommendation system. This embedding model used the sequence of products which are generated by user when visit website so, the product embedding can capture the relation between products in term of similar characteristic and user preference.
3. This framework can be applied with other industries such as retail industry.

3.1 Research Methodology

1. Study the related works and related theories
2. Data exploration
3. Data Preprocessing
4. Implement baseline models
5. Design model
6. Implement the proposed model
7. Result analysis and tuning

3.2 Research Paper

"Generating Realistic Users Using Generative Adversarial Network with Recommendation-based Embedding" (Awaiting for reply from IEEE Access)

For the remaining parts, in chapter II, we detail the related theories which consists of five topics: embedding, neural network, distance metric, deep generative model, and Gumbel-Softmax distribution. Chapter III covers literature review. We discuss related works contributed to product embedding and new approaches for generating realistic samples. The next chapter is methodology which is described the dataset and our approach. This approach is divided to two parts: product embedding model and generation model. The product embedding model is used to embed the product features into the embedding vector which capture the relationship between products. The generation model is used to generate realistic users given on product. Chapter V is experimental result. This part covers baseline approaches for both product embedding model and generation model, evaluation metrics, and model performance. The last chapter is conclusion and future work.

CHAPTER II

RELATED THEORIES

In this section, we will explain the related theories divided into 5 topics as follows: embedding, neural network, distance metric, deep generative model and Gumbel-Softmax distribution.

2.1 Embedding

To transform the categorical variable into continuous vector, there are 2 main methods: one-hot encoding and embedding. The one-hot encoding is a simple mapping the categorical variable to a binary vector with length equal to the number of categories. This vector is assigned value of 0 except the one index which is assigned with 1 to identify the value of that categorical variable. The limitation of one-hot encoding is that if we have 10,000 categories, the vector to represent this will be a 10,000-dimensional vector for each category. This vector is sparse and computationally expensive.

An embedding is a mapping of a categorical variable into low-dimensional and continuous vector. An ideal embedding can capture the semantics of the input by mapping the similar inputs close together in the embedding space. As the Figure 1, if the words are similar, their embedding vectors are closer. Thus, adding and subtracting embeddings can be interpreted. An embedding is considered as a dense and compact representation of feature vector.

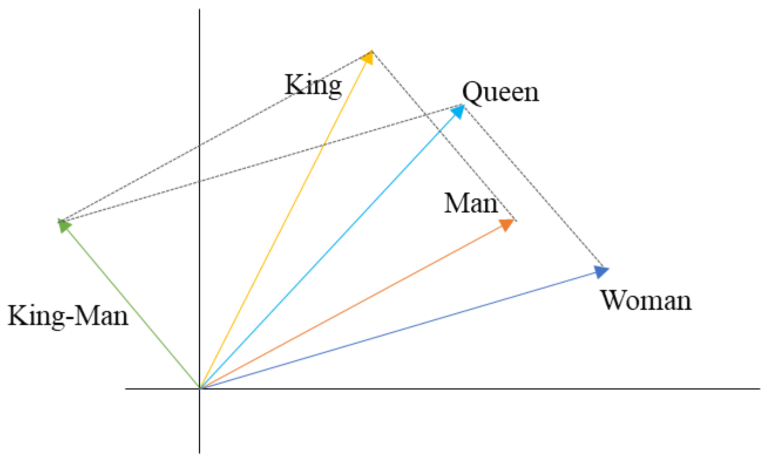

Figure 1 Example of word embedding vector in embedding space

2.2 Neural Network

A neural network is a model that imitates the human brain operation to recognize patterns based on training data to predict the unseen data.

2.2.1 Perceptron

Perceptron is a simplest type of neural network. This is a model of single neuron for binary classification.

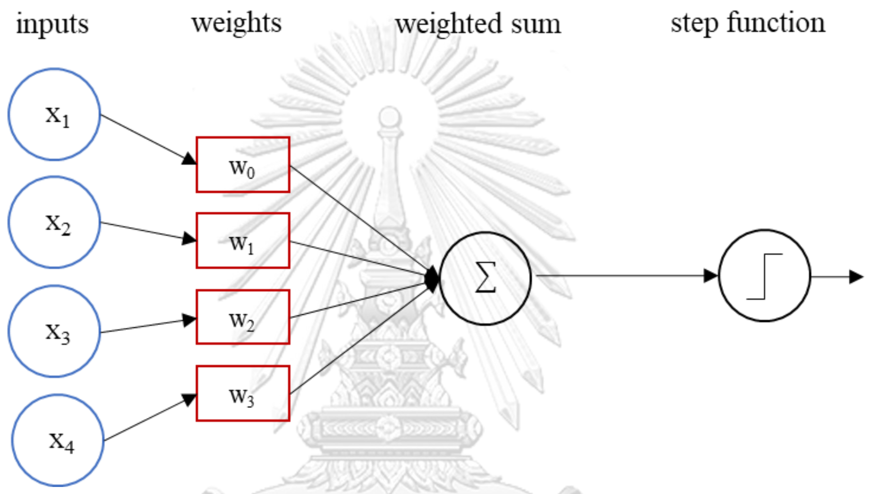

Figure 2 Perceptron's component

As Figure 2, the components of a perceptron are enumerated as:

- **Input layer:** the input of a perceptron is numerical vector of all features which are denoted as $[x_1, x_2, \dots, x_n]$ where n is the number of input features.
- **Weight:** each input has an associated weight which is learned and adjusted when training model. The weight of each input is relative to the important to other inputs. w_0 is a bias that is a constant. The bias value allows to shift the decision boundary for better quality of model and does not depend on any input value.
- **Weighted summation:** This node applies a function to weighted sum of input value (x_i) with associated weight (w_i) of each input. The formula is denoted as:

$$y = \sum_{i=1}^{n} w_i x_i \tag{1}$$

This output is transferred to the next layer.

- **Step function or Activation function:** this function is mathematical gate between the input and output to the next layer. This function uses to determines whether each neuron should be activated or not based on the relevant for prediction and also uses to normalize the scale of output. The non-linear activation function can help the network learn complex data. The example of activation function is shown below:
    - Sigmoid function: this function normalizes the output value between 0 and 1 and is calculated as the below formula:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2}$$

    - Softmax function: the output values bound is between 0 and 1. This is used as the probability of each class. Thus, the summation of this value is equal to 1 as the below formula:

$$f(z)_j = \frac{e^{z_j}}{\sum_{j=1}^{k} e^{z_j}} \tag{3}$$

    - Tanh function: the output values bound is between -1 and 1.

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{4}$$

    - ReLU (Rectified Linear Unit): this function is computationally efficient but when inputs approach zero or negative, the gradient becomes zero. That makes the network cannot perform backpropagation.

$$R(z) = \max(0, z) \tag{5}$$

    - Leaky ReLU: This function fixes the problem of ReLU when the inputs are zero or negative. This has a small positive slope in the negative area for backpropagation even negative input value.

$$R(z) = \max(0.1 \times z, z) \tag{6}$$

To find the optimal weight of each node, the model learns from the training data to adjust the weight as the following equation:

$$w_i \leftarrow w_i + \Delta w_i \tag{7}$$

$$\Delta w_i = \propto (\hat{y} - y)x_i \tag{8}$$

where y is actual value, $\hat{y}$ is the output value of model, and $\propto$ is learning rate that is a hyperparameter to control the step size for updating the weight.

2.2.2 Feedforward Neural Network

Feedforward neural network is the multi-layer perceptron or fully connected network that the connections between node do not form a loop. In Figure 3, the flow of information is forward from the input nodes and then feed to the next layer, and finally to the output layer. Thus, each node in each layer connects with every node in the following layer by weight (w_{ij}).

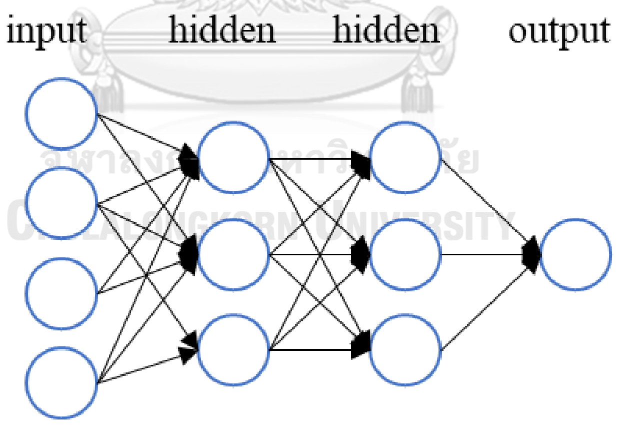

Figure 3 Feedforward neural network's operation

2.2.3 Recurrent Neural Network (RNN)

Recurrent neural network is a network for sequential data. Instead of considering only current input like feedforward neural network, this model takes both current input and output from previous state as inputs for current state. The main problem is vanishing and exploding gradient during training model which always happen when the network must learn the long data sequences. The value of gradient is smaller and may become to zero which means that model does not learn anything. To handle this problem, Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) were proposed.

2.2.1.1 Long Short Term Memory (LSTM)

The core concepts are the cell state that transfers the relevant information to the sequence and gates that decided which information is important to keep or forget. This model consists of three gates: forget gate, input gate and output gate.

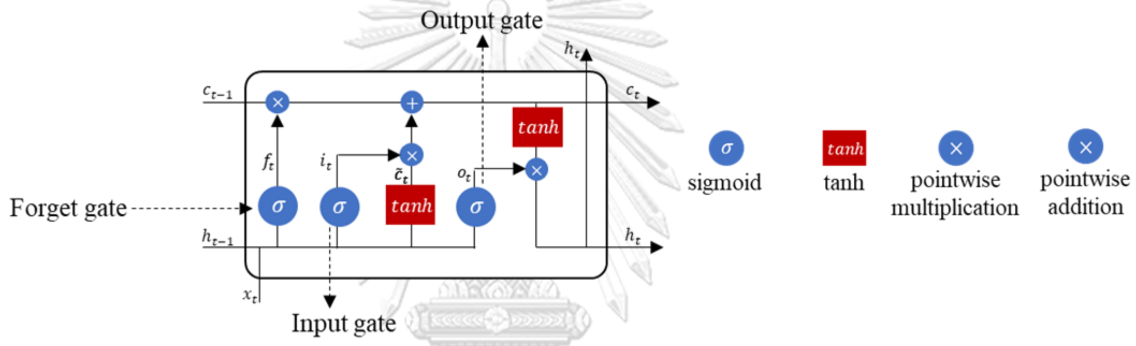

Figure 4 LSTM's operation

- Forget gate uses a sigmoid function to determine what information is relevant to keep from the previous state. If the value is close to 1, the information is relevant to keep.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{9}$$

- Input gate is a sigmoid function to determine what information is relevant to add from the current state. A tanh layer creates a vector of new candidate values $(\tilde{C}_t)$.

$$i_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_i) \tag{10}$$

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{11}$$

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{12}$$

- Output gate is a sigmoid function to determine what information is relevant to send to the next hidden state.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{13}$$

$$h_t = o_t \cdot \tanh(C_t) \tag{14}$$

2.2.1.2 Gated Recurrent Unit (GRU)

This network is similar to LSTM, but it is faster than LSTM. Instead of using the cell state to transfer information, GRU uses the hidden state with two gates: reset gate and update gate.

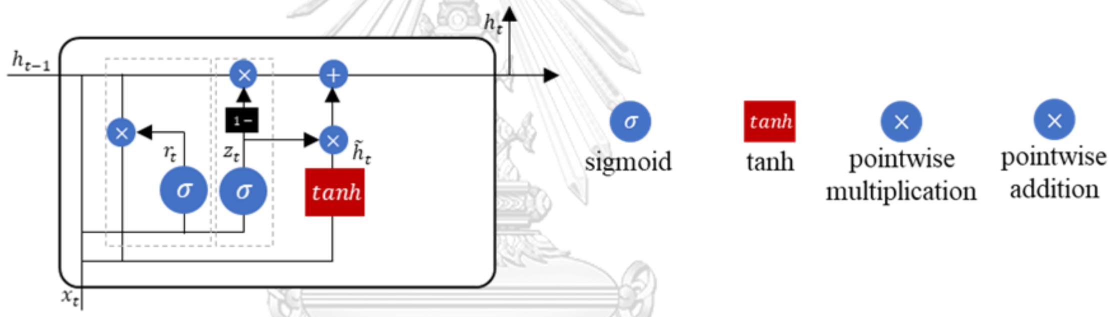

Figure 5 An example of GRU's operation

- Reset gate is used to decide how much the previous information to forget.

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{15}$$

- Update gate is used to determine what previous information is relevant to keep and what new information is relevant to add to the current state.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{16}$$

$$\tilde{h}_t = tanh(W \cdot [r_t \times h_{t-1}, x_t]) \tag{17}$$

$$h_t = (1 - z_t) \times h_{t-1} + z_t \times \tilde{h}_t \tag{18}$$

Both LSTM and GRU have gates to regulate the flow of information, keep the relevant information and pass along with sequences to make predictions.

2.2.4 Optimization Algorithm

Optimization algorithm is to minimize or maximize an objective function such as error function, distance function, etc.

2.2.4.1 Stochastic Gradient Descent (SGD)

Gradient Descent is a convex function. This algorithm is an iterative method to find the optimal values of parameters for finding the minimum value of cost function. At each iteration, the algorithm calculates the gradient and update to the new weight as the following formula:

$$w_t = w_{t-1} - \alpha \frac{\partial J}{\partial w} \tag{19}$$

where w is weight of each node, α is learning rate, J is cost function and $\frac{\partial J}{\partial w}$ is gradient of cost function. Instead of using all data for calculating the gradient for each iteration, stochastic gradient descent randomly selects a sample to find out the gradient of the cost function at each iteration. SGD is usually noisier than Gradient Descent algorithm, but the computation of SGD is lower than Gradient Descent to reach the optimal.

2.2.4.2 Root Mean Square Propagation (RMSProp)

This algorithm is similar to the gradient descent algorithm with momentum that uses the exponentially weighted average of the gradient for fixing the local optima problem. RMSProp divides the learning rate by an exponentially decaying average of squared gradients as the following equations:

$$g_t = \frac{\partial J_t}{\partial w} \tag{20}$$

$$MeanSquare_t = \gamma MeanSquare_{t-1} + (1 - \gamma) g_t^2 \tag{21}$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{MeanSquare_t}} g_t \tag{22}$$

where $MeanSquare_t$ is exponential average of squared gradients and γ is hyperparameters.

2.2.4.3 Adaptive Moment Estimation Algorithm (Adam)

This algorithm is an adaptive learning rate method to find the individual learning rate for each parameter. It combines both gradient descent with momentum and RMSProp as the following equations:

$$g_t = \frac{\partial J_t}{\partial w} \tag{23}$$

$$v_t = \beta v_{t-1} + (1-\beta) g_t \tag{24}$$

$$MeanSquare_t = \gamma MeanSquare_{t-1} + (1-\gamma) g_t^2 \tag{25}$$

$$w_t = w_{t-1} - \alpha \frac{v_t}{\sqrt{MeanSquare_t}} g_t \tag{26}$$

2.3 Distance Metric

To measure the difference between two probability distributions, there are three common measurements used in generative model.

2.3.1 Kullback-Leibler (KL) Divergence

KL divergence measures how much information we lose when we choose an approximated distribution. The formula is defined as:

$$D_{KL}(P||\, Q) = \sum_{x=1}^{n} P(x) log \frac{P(x)}{Q(x)} \tag{27}$$

Where P is the real data distribution and Q is the one estimated from the model. KL Divergence is not symmetric $(D_{KL}(P||\, Q) \neq D_{KL}(Q||\, P))$.

2.3.2 Jensen-Shannon (JS) Divergence

Because KL Divergence is asymmetric, JS Divergence is a symmetric version and smooth as the following formula:

$$D_{JS}(P||\,Q) = \frac{1}{2}D_{KL}(P||\frac{P+Q}{2}) + \frac{1}{2}D_{KL}(Q||\frac{P+Q}{2}) \tag{28}$$

Instead of distance between probability distributions of each other, this is an average of distance between their probability distributions and average of them.

2.3.3 Earth-Mover Distance (EMD) or Wasserstein Distance

There is gradient vanishing problem for KL Divergence and JS Divergence when the distribution Q far away from the distribution P. The Wasserstein distance has a smoother gradient. It is the minimum cost of transporting in converting the data distribution Q to the data distribution P as the following formula:

$$W(P,Q) = \inf_{\gamma\in\Pi(P,Q)} E_{(x,y)\sim\gamma}\,[\parallel x-y \parallel] \tag{29}$$

Where γ is a joint probability distribution. The Figure 6 shows examples of EMD calculation. The yellow graph is the distribution of P and the green graph is the distribution of Q. The minimum cost to transform distribution P to distribution Q is 5. At step [1], the cost of transforming P_1 to Q_1 is moving 2 boxes from P_1 to P_2. The cost of transforming P_2 to Q_2 is 2. The last one is moving 1 box from P_4 to P_3. Thus, the total cost is summation of all moving.

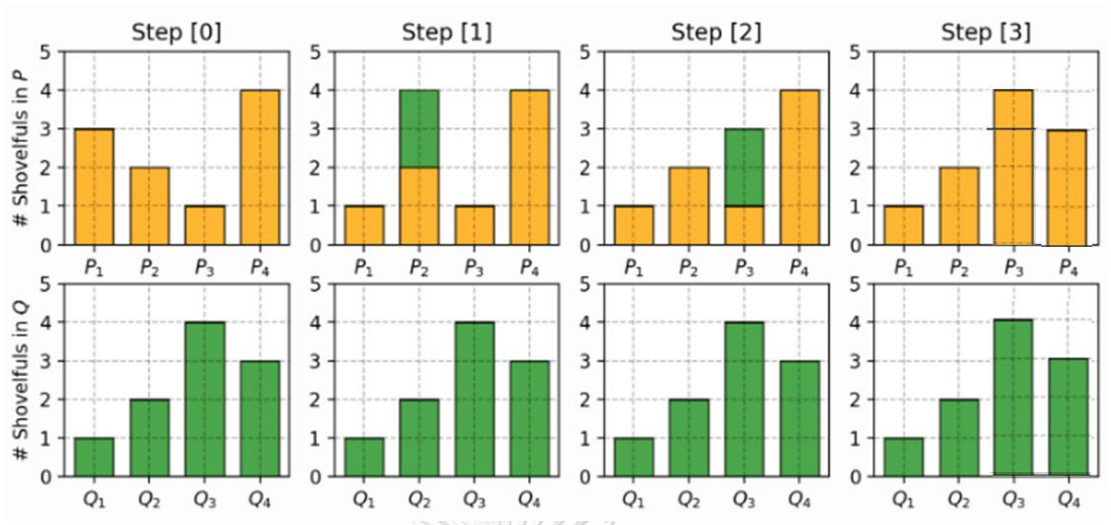

Figure 6 Example of Wasserstein distance calculation modified from figure 7 [17]

2.4 Deep Generative Model

For deep generative model as the Figure 1, this model learns via maximum likelihood estimation. Models of the left branch construct an explicit density and maximize their likelihood. Variational Autoencoder (VAE) is one of explicit approaches by making variational approximations. While Generative Adversarial Networks (GAN) is an implicit approach which directly sample from the distribution.

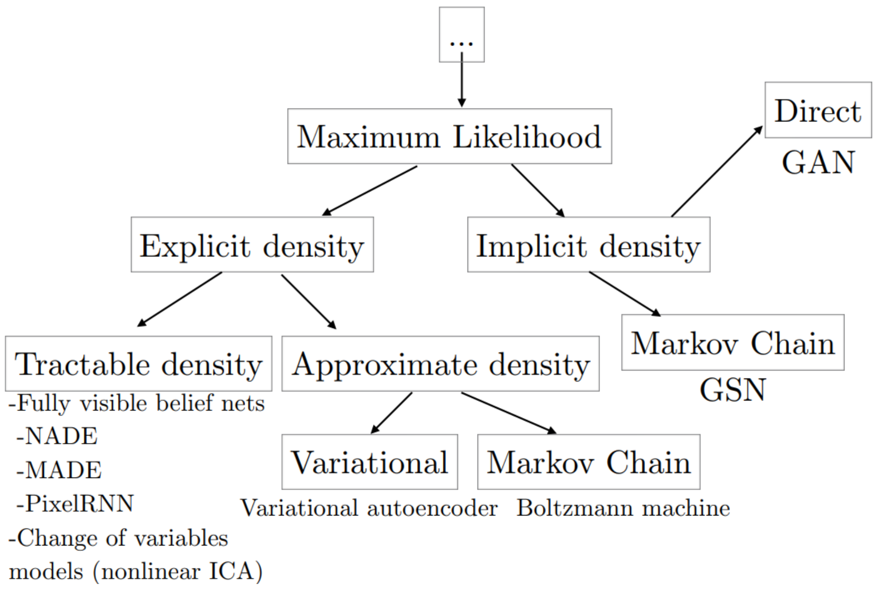

Figure 7 Deep generative model (Reference from figure 9 [18]*)*

2.4.1 Variational Autoencoder (VAE)

Standard Autoencoder [19] is a feed forward neural network for unsupervised learning. The common usage is dimensionality reduction, feature selection and feature extraction. The concept of model is learning to compress input (x) into a latent vector that preserve the relevant information and distribution of input. The output of model is the reconstructed input that is similar to the original input. As Figure 8, the autoencoder consists of 2 neural networks: encoder and decoder.

- The encoder is a neural network that takes input (x) and compress it to latent vector or dense representation (z) of data (x). This is called inference network which parameterizes the approximate posterior of the latent variables z. The output is the distribution $p_{\emptyset}(z|x)$.
- The decoder is a neural network that takes the latent representation (z) and learn to reconstruct original input. This is called generative network that outputs the likelihood distribution $p_{\theta}(x|z)$.

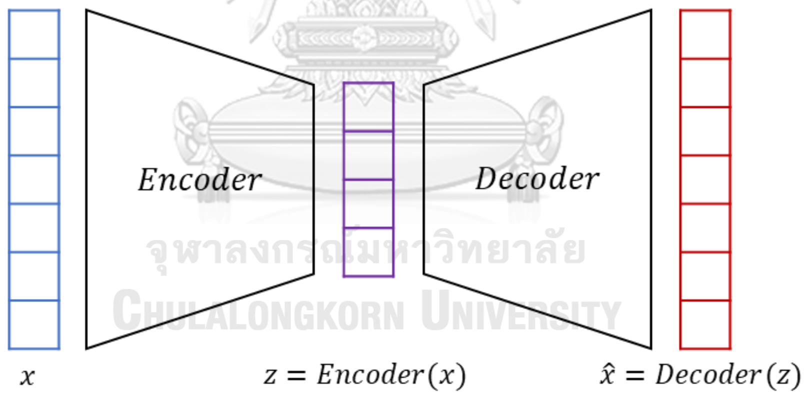

Figure 8 Autoencoder's architecture

The loss function is to minimize the mean square error (MSE) or cross-entropy between original inputs (x) and outputs $(\hat{x})$ of the model. Thus, the representation contains the relevant information for decoder. The limitation of autoencoder is new data generation. After we train model, we get the encoder and decoder, but cannot generate any new data from this model.

Variational Autoencoder (VAE) [20] resembles autoencoder, but VAE is a deep generative model to generate new sample data. In order to use the decoder for new data generation, VAE generate the variations on input before passes to the decoder for new data generation.

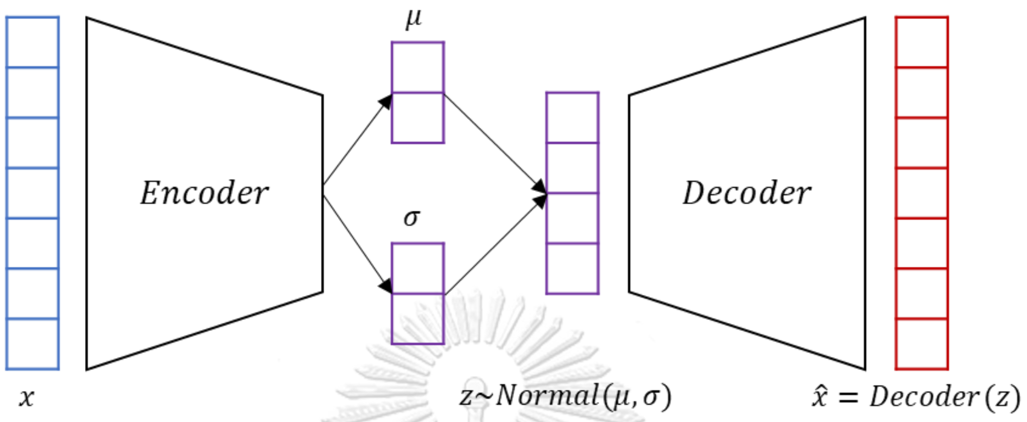

Figure 9 Variational Autoencoder's architecture

As Figure 9, VAE consists of 2 neural networks: an encoder and a decoder which are like the autoencoder. To generate the variation for new data generation, VAE randomly sample latent representations following a Gaussian distribution with mean (μ) and standard deviation (σ) that are outputs from the encoder. The encoder outputs the latent distribution instead of latent representation. The mean vector controls where the representation of an input should be centered, and the standard deviation controls the variation from the mean of the representation. Thus, the decoder can learn not only single point in latent space which refers to an input but also all nearby points in latent space are referred to the same input. Moreover, the latent space is stochastic node because of sampling. In model training, we cannot calculate the gradient via stochastic node. Thus, the authors in [20] proposed "Reparameterization trick". Instead of sampling z from Gaussian distribution with μ and σ, this trick converts the sampling to calculate z from $\mu + (\sigma^2 \times Normal(0,1))$. This trick allows gradient to flow via μ and σ nodes to update for optimal weight.

The loss function of VAE is to minimize the Evidence Lower Bound (ELBO) as the following formula:

$$ELBO(\theta, \emptyset) = E[log\, p_{\emptyset}(x|z)] - D_{KL}[p_{\theta}(z|x)||p(z)] \tag{30}$$

Where $p_{\emptyset}(x|z)$ is the probability distribution of decoder's output, $p_{\theta}(z|x)$ is the probability distribution of decoder's output and $p(z)$ is the probability distribution of latent representation which is a standard normal distribution. The first term $E[log\, p_{\emptyset}(x|z)]$ is the reconstruction loss to encourage the decoder learns to reconstruct the new data over the representation. The second term $D_{KL}[p_{\theta}(z|x)||p(z)]$ is the regularization to make the distribution returned from the encoder closes to the standard normal distribution. Thus, this term is measured by KL divergence metric to calculate the distance between the encoder's distribution and the prior distribution.

2.4.2 Generative Adversarial Network (GAN)

Generative Adversarial Network [5] is a deep generative model which can generate realistic samples that are similar to the training data. The model is composed of a generator (G) and a discriminator (D) as Figure 10.

- The generator takes a random noise input that is sampled from standard normal distribution or standard uniform distribution and learns to generate new realistic samples.
- The discriminator decides whether a sample is from real or generated distribution.

The training of GAN is done as a minimax game where the discriminator tries to maximize the likelihood to recognize real samples as real and generated samples as fake while the generator would like to minimize it.

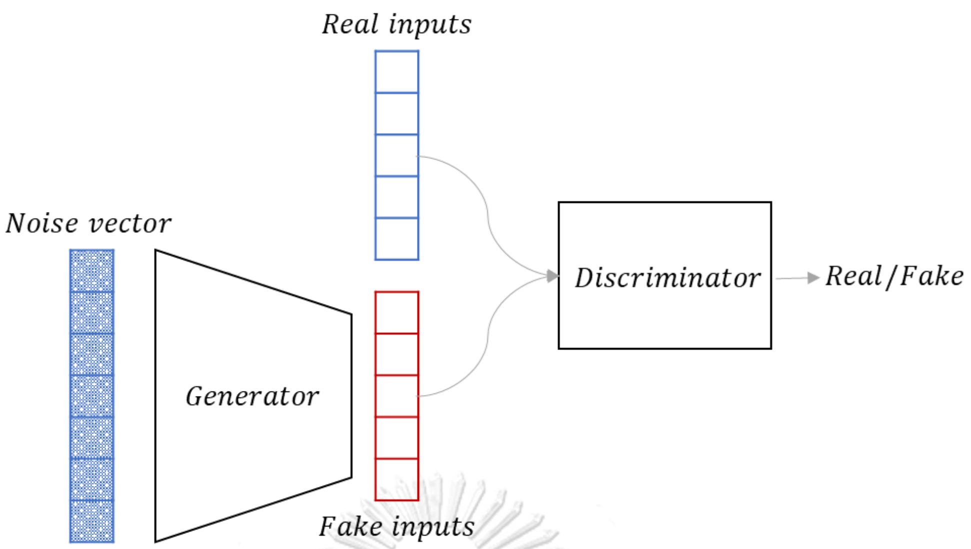

Figure 10 Generative Adversarial Network's architecture

GAN aims to achieve an equilibrium between the generator and the discriminator. The loss function of the discriminator is defined as:

$$J^D = -\frac{1}{2}E_{x\sim p_{data}}[logD(x)] - \frac{1}{2}E_{x\sim p_z}[log(1 - G(x))] \tag{31}$$

Where p_{data} is real data distribution, p_z is normal distribution (0, 1) or uniform distribution (0, 1). This is an entropy between real and generated distribution. For the generator, the loss function is defined as:

$$J^G = -J^D \tag{32}$$

There are many problems in training GAN such as non-convergence, diminished gradient, mode collapse when the generator produces limited varieties of samples, etc. One of them is vanishing gradients. The loss function of GAN is the average of distance between the real or generated distribution and the average of them. This is equivalent to the JS divergence. When the generated distribution is far away from the real distribution, the gradient for the generator will vanish. Moreover, there is no guarantee that JS divergence will be continuous and differentiable everywhere [21].

Wasserstein Generative Adversarial Networks (WGAN) [21] was proposed to improve GAN by using new loss function. They use Wasserstein distance that has a smoother gradient than JS divergence as the loss function and rename the discriminator to the critic. The output of critic is a scalar score without sigmoid function to measure how real the input sample are. Computing the Wasserstein distance is hard to control. Thus, the approximation using Kantorovich-Rubinstein duality is defined as below:

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)] \tag{33}$$

Where P_r is the real data distribution, P_g is the generator distribution that tries to approximate P_r and f is a 1- Lipschitz function following the constraint: $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$. Because of the constraint of Lipschitz continuity, we need to use weight clamping. The weights (w) of the critic must be constrained within a certain range [-c, c] by clipping w after every update to w. Thus, c is a hyperparameter that we need to tune. If the $W(P_r, P_g)$ is small, the output of the generator is close to the real sample.

The difficulty in WGAN is tuning the bounds in weight clipping. If the bound is large, it can take a long time for the weights to converge. If the bound is small, it can lead to vanishing gradients when the network has many layers. To alleviate this issue, WGAN-GP [22] was proposed. Instead of the weight clipping, the authors added a gradient penalty term into the loss function to enforce the Lipschitz constraint during the training phase. The loss function of the both discriminator (J_W^D) and generator (J_W^G) are defined as:

$$J_w^D = -E_{x \sim p_g}[D(x)] + E_{x \sim p_r}[D(x)] - \lambda E_{x \sim p_{x'}}[\|\nabla_x D(x)\|_2 - 1]^2 \tag{34}$$

$$J_w^G = -J_w^D \tag{35}$$

Where the last term is gradient penalty. $p_{x'}$ is a uniform distribution along the straight lines between pairs of points which are sampled from the data distribution p_r and the generated data distribution p_g.

2.5 Gumbel-Softmax Distribution

Because of the difficulty of training stochastic networks with discrete variables, the authors in [14, 15] proposed Gumbel-softmax trick which can be used to backpropagate through the softmax. We should begin with the Gumbel distribution, the Gumbel-Max trick and the Gumbel-Softmax trick.

2.5.1 Gumbel Distribution

The Gumbel distribution is proposed in [23] by Gumbel to model the extreme value distribution which has two parameters: μ and β. The standard Gumbel distribution is the case where $\mu = 0$ and $\beta = 1$ as the Figure 11. This distribution can be drawn by inverse transform sampling as the following formula:

$$g = -log(-log(u)) \tag{36}$$

where u is a uniform distribution on the interval (0, 1).

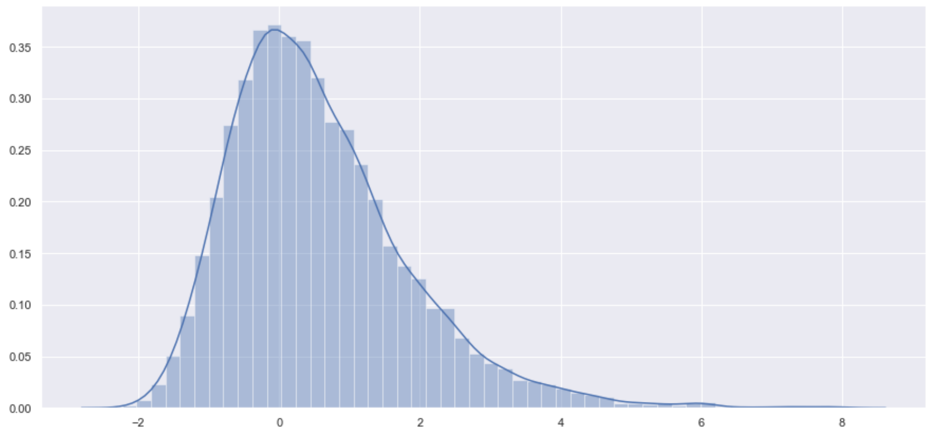

Figure 11 The standard Gumbel distribution

2.5.2 Gumbel-Max trick

The Gumbel-Max trick was proposed in [24] to draw a sample from a discrete distribution as the below formula:

$$z = onehot(argmax_i[g_i + log\pi_i]) \tag{37}$$

Where $onehot$ is a function to encode categorical variable into binary vector which is sparse matrix, $argmax$ is a function to return the maximized value, g_i is a noise which is drawn from the standard Gumbel distribution or $Gumbel(0,1)$, and π_i is class probabilities.

2.5.3 Gumbel-Softmax trick

Because of the Gumbel-Max trick is not differentiable because of argmax layer, the Gumbel-Softmax trick was proposed in [14, 15] by using the softmax function to approximate the argmax as the following equation:

$$y_i = \frac{exp(log(\pi_i) + g_i)/\tau}{\sum_{j=1}^{k} exp(log(\pi_i) + g_i)/\tau} \tag{38}$$

Where τ is the softmax temperature parameter. When τ approaches zero, the expected value of the Gumbel-Softmax distribution is identical to a discrete distribution. At higher τ, the expected value converses to a uniform distribution. The Gumbel-Softmax function yields a smooth gradient at high τ, but can become unstable at low τ. At high τ can be used at the start of the training and anneal to a small but non-zero value.

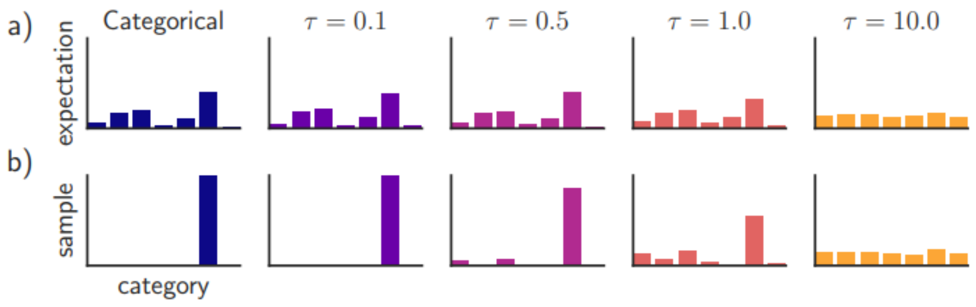

Figure 12 The Gumbel-Softmax distribution when use the high temperature (τ) and low temperature (τ) (Reference from figure 1 [14]*)*

2.5.4 Straight-Through Gumbel Estimator

For scenarios that requires sampling of discrete values such as ones in GAN, the authors in [14] proposed the Straight-Through (ST) Gumbel Estimator. The forward pass is done by normal sampling, but the backward pass is done by backpropagating the Gumbel-Softmax to approximate the gradient which is shown in Figure 13.

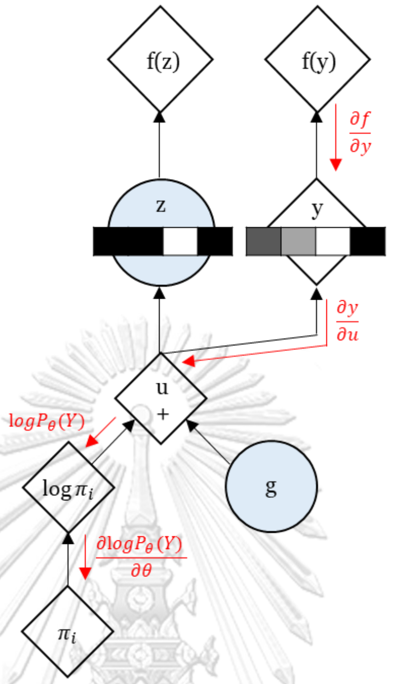
f(z)
f(y)
$\frac{\partial f}{\partial y}$
z
y
$\frac{\partial y}{\partial u}$
u
+
$\log P_\theta(Y)$
$\log \pi_i$
g
$\frac{\partial \log P_\theta(Y)}{\partial \theta}$
π_i

Figure 13 The Straight-Through Gumbel Estimator's operation

CHAPTER III

LITERRATURE REVIEW

In this work, we bring together elements from related works contributed to product embedding and generating new realistic samples approaches. We briefly review them below.

3.1 Product Embedding

An embedding can be considered as a dense and compact representation of feature vectors. Good embedding should have the property that if the features are similar in some sense, their embedding vectors are closer. Word2Vec, which was proposed in [25], is a popular embedding technique for textual data. Two words are considered similar semantically by their surrounding words and co-occurrence. The authors proposed two methods to learn Word2Vec embedding: Continuous Bag of Words (CBOW), which predict the current word based on surrounding words, and Skip-gram, which predict context words given the current word.

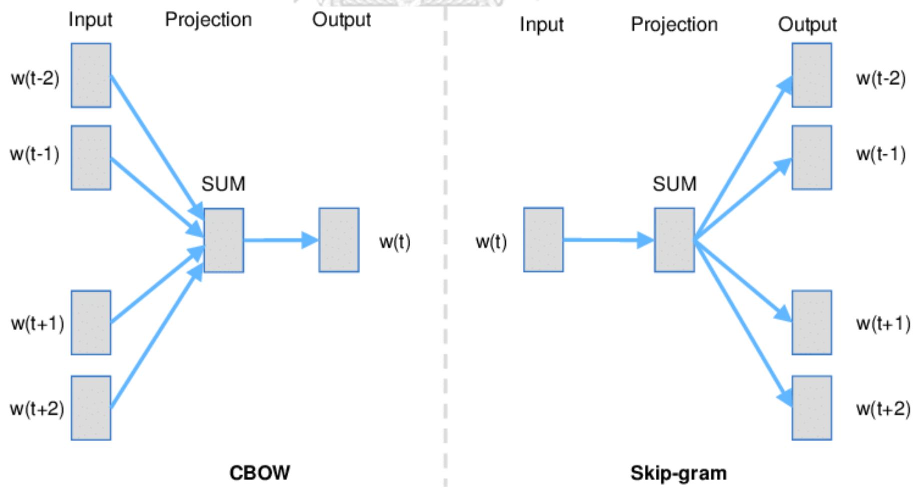

Figure 14 The Continuous Bag of Words (CBOW) and the Skip-gram model (Reference from figure 3 [26])

For recommendation system, item embedding is a continuous vector which represents item and tries to capture the relationship of items. In [27], the authors proposed Item2Vec by using the Word2Vec framework. They assumed that the items which share the same basket are similar in some sense regardless of the order that user generates. They predicted a item based on other items in the same basket. The authors in [4] applied item embedding to improve the session-based recommendation task. They embedded an item description to its embedding by using the Word2Vec and the GloVe method as input to predict next clicked item in an e-commerce website. In [28], the authors want to represent the relation between users and content for news recommendation. They generated user representation by using a recurrent neural network (RNN) while the content embedding was learned via denoising autoencoder. They shown that the click-through rate improved by 23% and the total duration improved by 10% over not using embeddings.

3.2 Generating new realistic samples approaches

Most works for generating new realistic samples are contributed by Variational Autoencoder (VAE) and Generative Adversarial Network (GAN). In 2013, VAE was proposed in [20]. This model is a kind of Autoencoder that can learn the distribution of the data. VAE can be used to generate novel data samples by sampling from the learned distribution. To control the data generation on VAE when we need to generate some specific data, Conditional Variational Autoencoder (C-VAE) [16] was proposed in the following year by adding conditional input to the both encoder and decoder networks. The decoder can generate new sample from based on the additional information.

In application domains, VAE was used in natural language processing. [29, 30] applied VAE with sequential model to the task of dialogue response generation by learn the distribution of conversation. In [31], the authors used VAE to the latent image features and then generate related captions of that image. JointVAE was proposed in [32] to learn both continuous and discrete representations because the original VAE can learn the features with Gaussian distribution. So, this work proposed to apply Gumbel-Softmax trick for discrete features.

For recommender system, the authors in [33] applied VAE to generate new items which maximally satisfy the preference of a group of users. They learned the share latent representation between user and item features from user-item ratings. Embedding for product recommendations can be generated through weighted maximum coverage in a greedy manner. The item decoder maps these latent representations to item features of novel items.

However, the author in [18] said that VAE usually fails to capture multimodal distributions and usually generate lower quality outputs due to the gap between the lower bound of approximate posterior distribution and the true data distribution. On the other hand, no variational bound is needed in GAN. That makes GAN can generate better results.

In 2014, GAN was proposed in [5] by Ian J. Goodfellow . This model is a deep generative model to generate new realistic samples. The generator learns how to generate realistic outputs which can fool the discriminator. The discriminator learns to discriminate which input come from real data or generated data. In [34], the authors proposed a conditional generative adversarial network (CGAN) to control the specific output of generator. They add some information (y) such as class label and other modalities into both the generator and the discriminator as the Figure 15. The loss function is same as the original GAN. InfoGAN [35] is another work which conditioned the generation by adding the latent code to the generator. The discriminator predicts which input come from real or generated data and outputs the latent code. The loss function is added the mutual information between the latent code and the generator output as a regularization term.

GAN has been successfully applied in many application domains such as computer vision and natural language processing. In [10], the authors proposed CycleGAN to transform images from one domain to another domain. The authors in [7, 9] applied VAE and GAN to repair and fill the missing parts of images, a tasked is called image inpainting. To improve the quality of generated samples, there are many works that combine VAE and GAN, especially for computer vision [6, 8, 36, 37]. VAE is an excellent generative model for learning representations but generates blurry outputs, while GAN generates sharp outputs but cannot explicitly learn the embeddings like VAEs. In [6], the authors used two generative models to capture the latent spaces of hand poses and depth images for 3D hand pose estimation. VAE embedded features to the share latent representation and GAN's generator generated the 3D hand pose by latent representation.

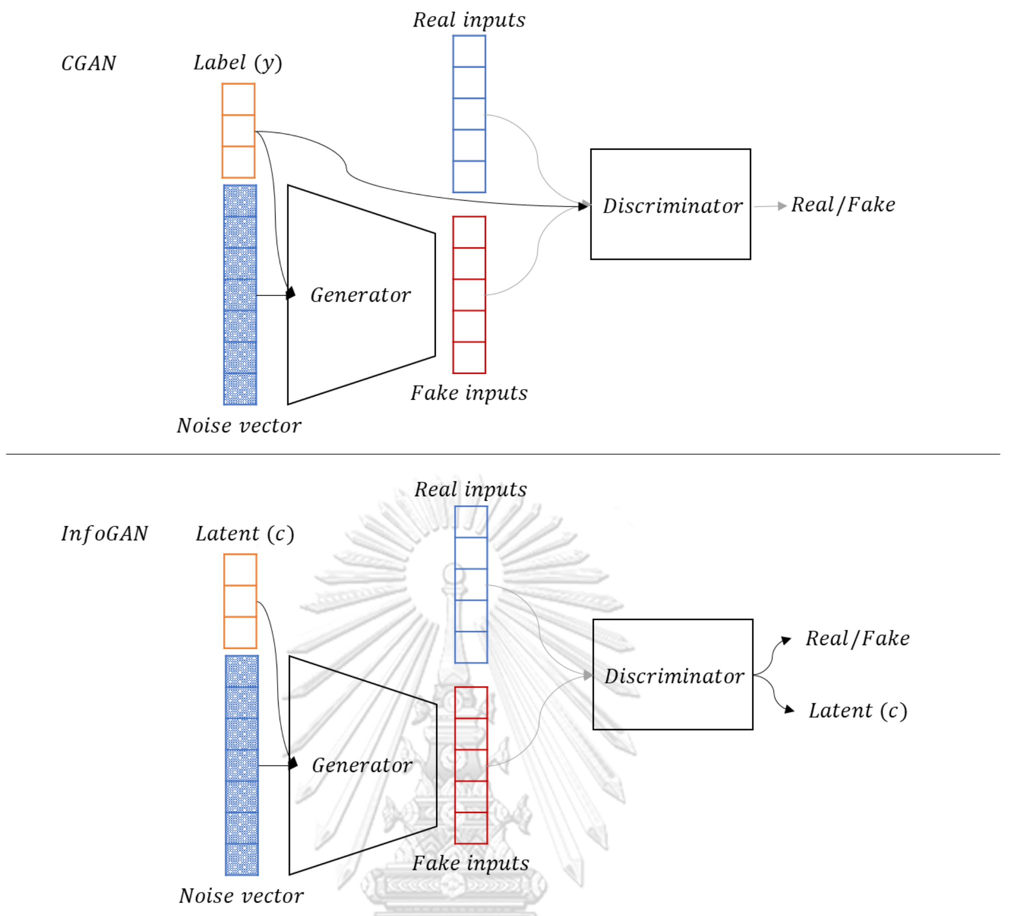

Figure 15 Architecture of CGAN and InfoGAN

For NLP, the limitation of GAN is in generating discrete outputs such as text because the gradient cannot be back propagated through the argmax function used to generate a discrete output. REINFORCE, a technique used in the reinforcement learning literature, can be used to circumvent this issue [12]. However, it can lead to slow convergence and training instability. In [11], the authors applied Gumbel-Softmax trick which was proposed in [14] for text generation to handle discrete outputs. In business applications, [13] was recently proposed to apply GAN with e-commerce data to generate the plausible orders related to a particular product in order to understand the characteristics of future orders. This model is called ec^{2}GAN. However, the output of the generator is order embedding. Thus, the authors need to train classifiers to extract the characteristics from generated order representation which makes the training process sophisticated and hard to maintain.

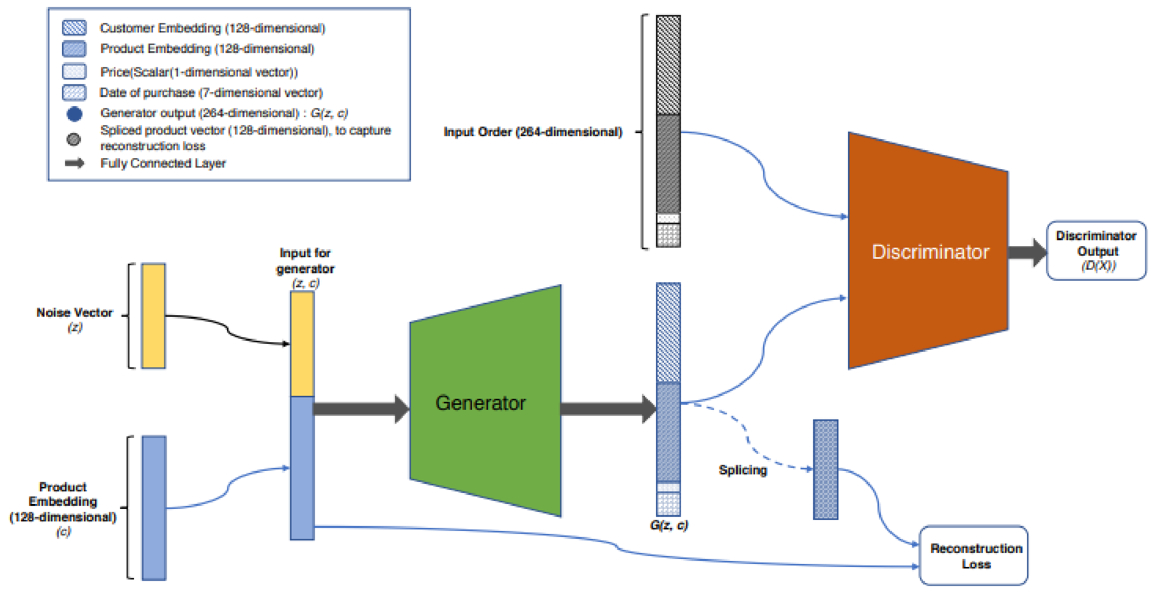
Customer Embedding (128-dimensional)
Product Embedding (128-dimensional)
Price(Scalar(1-dimensional vector))
Date of purchase (7-dimensional vector)
Generator output (264-dimensional) : G(z, c)
Spliced product vector (128-dimensional), to capture reconstruction loss
Fully Connected Layer
Input Order (264-dimensional)
Discriminator
Discriminator Output (D(X))
Input for generator (z, c)
Noise Vector (z)
Generator
Product Embedding (128-dimensional) (c)
Splicing
G(z, c)
Reconstruction Loss

Figure 16 The architecture of ec^{2}GAN

The key contributions of this work are enumerated below:

- We applied the conditional generative adversarial network with real-estate domain to generate realistic logs of user for new products before its release.
- We handled the limitation of discrete outputs generation by using the Straight-Through Gumbel Estimator instead of a two-step process as the prior work.
- We extracted the product embedding via the recommendation-based model. We trained a Gated Recurrent Unit (GRU) to predict the next product that user will click. Thus, the embedding can capture the similar product characteristics and also the similar preference of users.

CHAPTER IV

METHODOLOGY

In this section, we describe the details of the dataset and our system for generating user logs. Figure 17 shows an overview of our system. First, we embed the product features with some embedding model. Then, we feed the product embedding to the generation model which will output user logs. The logs can be analyzed to extract insight about the product used as the input.

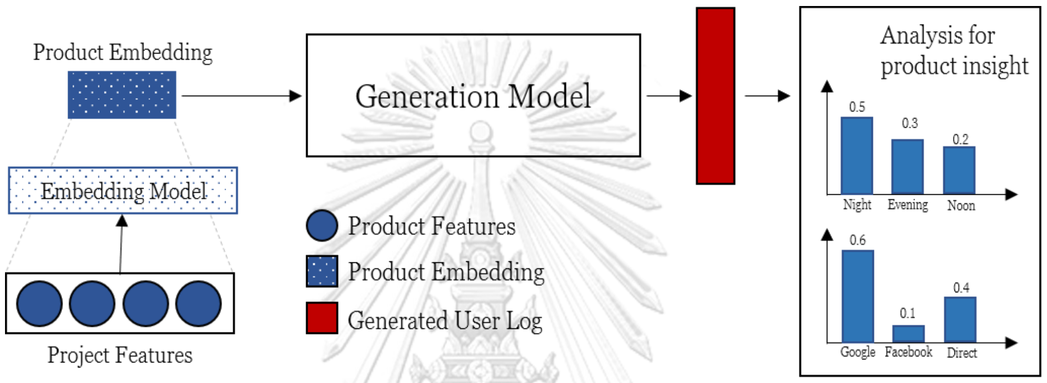

Figure 17 The overview of our system

For embedding model, we compare the performance of three approaches: using ULMFiT with product description, using Autoencoder with product features and using recommendation model with product features. For generation model, we also compare three approaches: nearest neighbor approach (NN), conditional variational autoencoder (CVAE) and conditional generative adversarial network (CGAN). Our proposed approach is using recommendation-based embedding with CGAN.

4.1 Dataset

Our web log data are from a real-estate search engine website (https://www.home.co.th) from January 2018 to February 2018. There are around 1.5 million records and 5,400 property projects. An example of the web log data is given in Table I. It consists of the user ID, the project that the user visited, the device that the user used to access the website, the agent or operation system, the referring page that the user was referred from, and the time of visit.

Table 1 An example of our web log data

User id	Project	Device	Agent	Referring page	Year	Month	Day	Hour
8bx-xx-xxx	5174	Mobile	iPhone	Google	2018	1	15	8
8bx-xx-xxx	5476	Mobile	iPhone	Google	2018	1	15	8
1ax-xx-xxx	7956	Desktop	Windows	Direct	2018	1	15	12
F7x-xx-xxx	3924	Mobile	Android	Others	2018	1	16	21

4.1.1 User features

The user features (U_i) are created from the web log which contain the follow features:

- Customer characteristics:
  - Device (Mobile, Desktop)
  - Agent or operation system (Android, iPhone, iPad, Macintosh, Windows, Others)
  - Customer segmentation (based on K-mean clustering)
- Channel:
  - Referring page (Google, Facebook, Direct, Others)
- Visit period:
  - Day of week (Mon, Tue, Wed, Thu, Fri, Sat, Sun)
  - Period (Morning, Afternoon, Evening, Night)

An example of features constructed from each log entry is shown in Table II.

Table 2 An example of user features that visited project in the website

Device	Agent	Referring page	Cluster	Weekday	Period
Mobile	iPhone	Google	2	Mon	Morning
Mobile	iPhone	Google	2	Mon	Night
Desktop	Windows	Direct	1	Thu	Afternoon
Mobile	Android	Others	35	Sat	Evening

To give a sense of our data, we show the histogram of each features in Figure 18. The users mostly use mobile to access website 60.6%. Most sessions were referred from Google. While the peak time period is during afternoon (13:00 – 18:58). The distribution of customer segmentation is highly imbalance. 28% of users is in customer group 13.

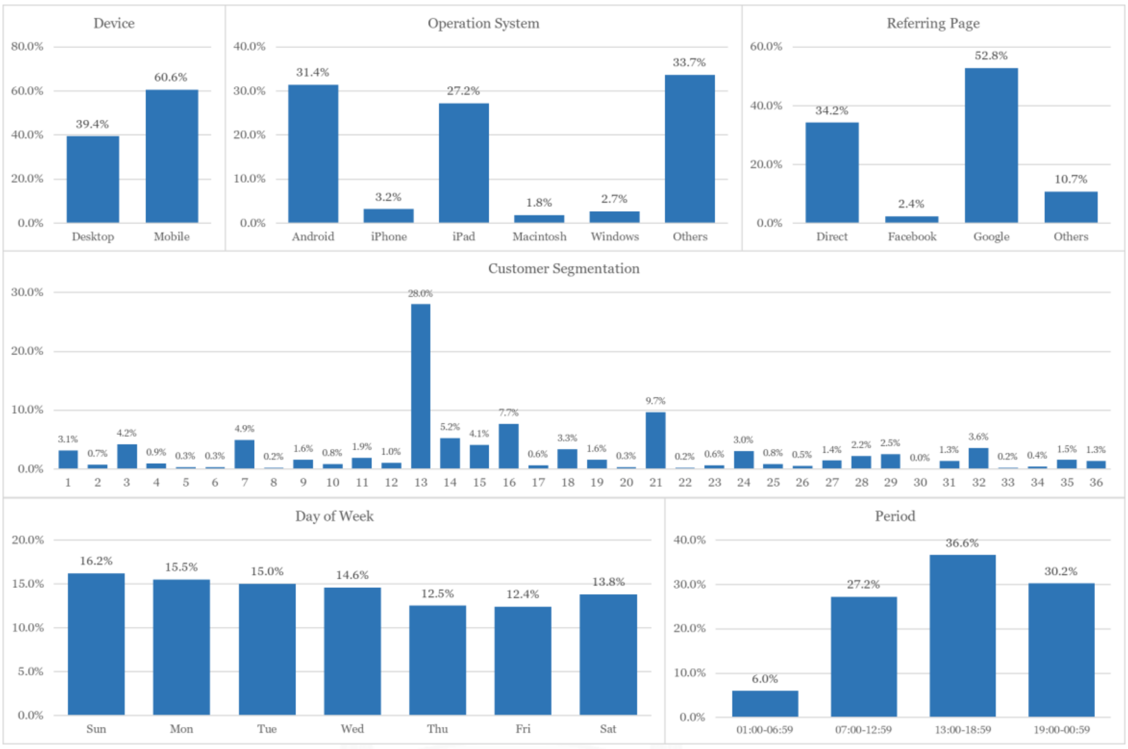

Figure 18 The distribution of number of projects per user which visited more than 1 project.

4.1.2 Product features

We also use product features (C_j) which capture the characteristics of each property projects. The product features include: the starting price, the location, the nearest train station, the latitude, the longitude, the area, the district, the project type, and the facilities. The product feature is used to condition the generation algorithm. We grouped features to their category as follow:

- Start period of project
- Location such as latitude, longitude, and district

- Transportation and landmark such as train station, express way, and supermarket
- Facility such as swimming pool, parking lots and gym
- Project type and style such as condominium, detached house, and home office

To reduce spurious information, the Figure 19 shows the distribution of number of users per project. Around 20% of projects were visited less than 50 users. Thus, we filtered out these projects that were visited less than 50 users which result in 4,876 projects remaining.

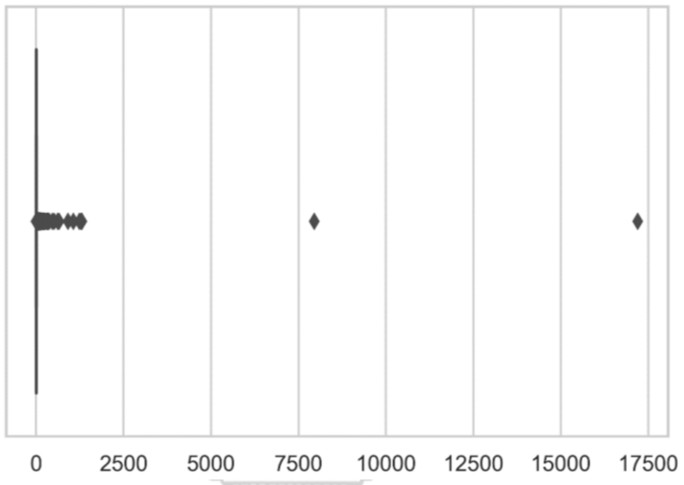

Figure 19 The distribution of number of users per project

To remove the users which can be bot, the Figure 20 shows the distribution of number of projects per user. At the 99.5 percentile of users, they visited over 30 projects within an hour. Thus, we filtered these users from our data before training model.

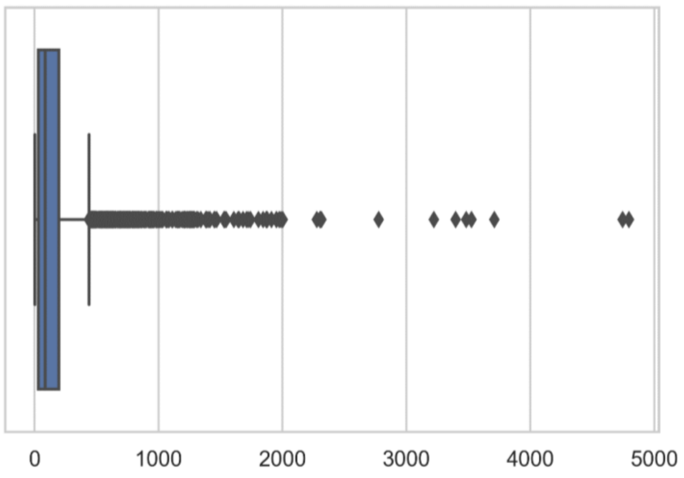

Figure 20 The distribution of number of projects per user

4.2 Product Embedding

One of the key components of our model is the product embedding. It aims to encapsulate each product's peculiarity so that the generator can have an easier time generating new visit logs.

We extract the embedding via a trained recommendation system. The goal is to capture additional properties of the product that are not captured in the pattern of how users explore the products. Our system is similar to the recurrent neural network-based recommender. Our model consists of two parts: the encoder and the predictor as shown in Figure 22. The recommendation model takes the sequence of products visited by each user as input and tries to predict the next product that the same user would visit. The sequences of product are represented by their product features. The product features are embedded into an embedding via a three fully connected layers. The embedding is fed to a Gated Recurrent Unit (GRU) which is used to predict the next product.

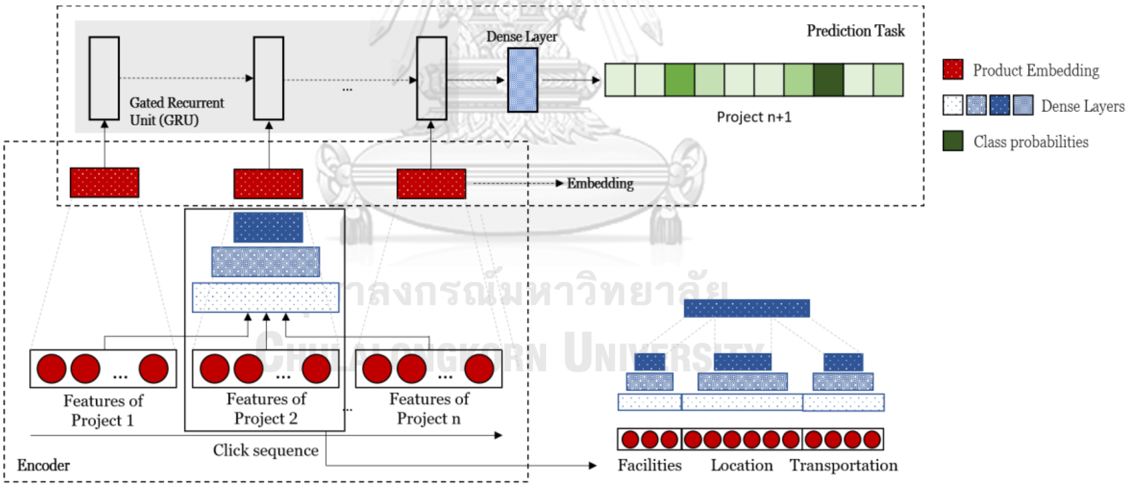

Figure 21 The recommendation system used to produce product embeddings

4.2.1 Encoder

The encoder is a concatenation of the fully connected networks of each product category feature. We group product features to category as we explained in the previous section and embed each category to its embedding before concatenating to final embedding of each product as you can see in the Figure 23.

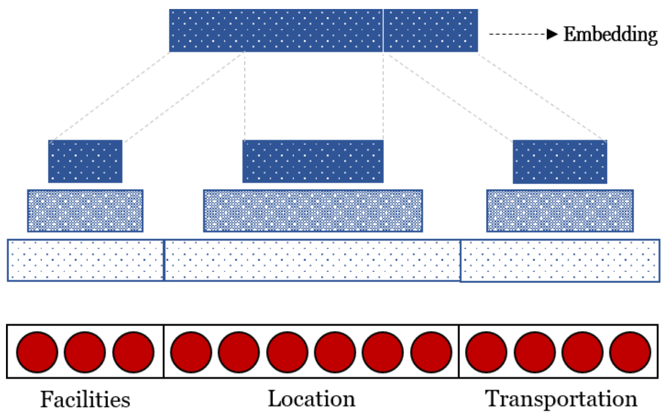

Figure 22 The encoder of each sequence

4.2.2 Prediction task

The prediction task is a Gated Recurrent Unit (GRU). The input is the sequence of product embedding which is generated by user when visited website. We pass this embedding to 3 dense layers with (128, 96, 64) neurons at each layer. The last layer is a dense with sigmoid function to predict the next product that users will visit.

After training model, the embedding vector from the encoder is the embedding for each product. This product embedding vector captures the relationship between projects which is not only the similar characteristics but also the similar user preference.

4.3 Generative Model

The key of our system is the generative model which takes in a product embedding and outputs user logs. Our GAN-based generator is shown in Figure 24. The model is similar to [13] which generated the orders of novel product in e-commerce website. However, their generated orders are embedding due to the limitation of GAN to handle discrete output generation, they need to train classifiers to extract the characteristics from generated representation. Our work uses the Straight-Through Gumbel Estimator [14] in order to deal with discrete generation.

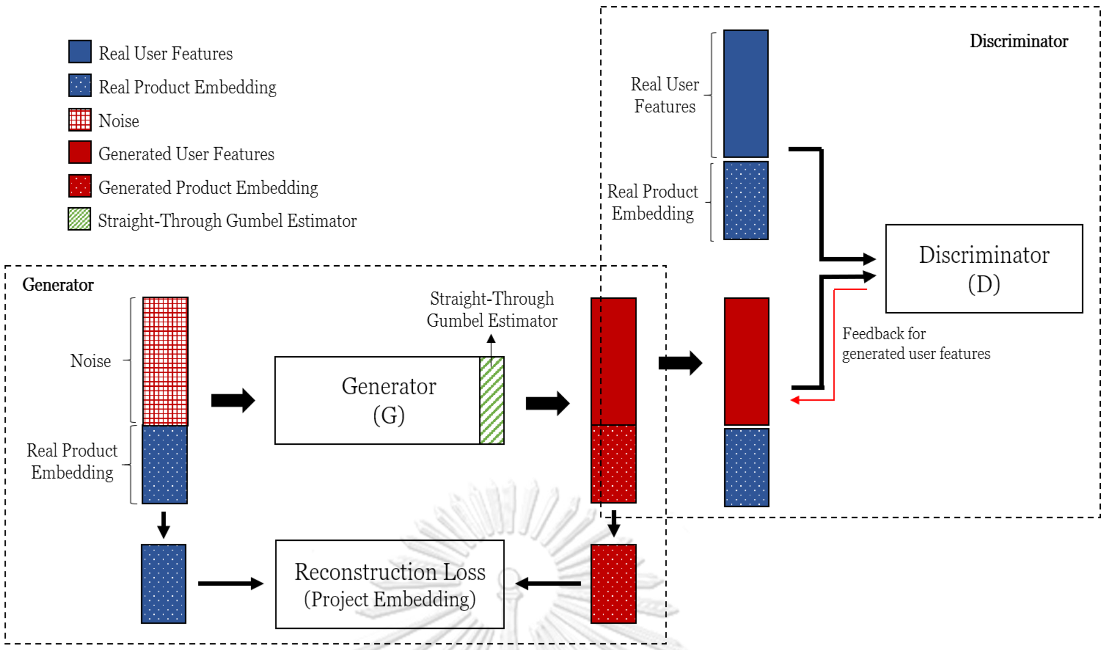

Figure 23 The GAN model for web log generation

4.3.1 Generator (G)

The generator generates fake user features conditioned on the product embedding. In other words, the generator will try to generate users that are likely to interact with the particular product. The generator is a fully connected network with three hidden layers and uses LeakyRelu as activation function at each layer. The last layer, we use the Straight-Through Gumbel estimator to generate the discrete outputs and the tanh function to generate the continuous outputs which were normalized to the range of [-1, 1].

4.3.2 Discriminator (D)

The discriminator takes the concatenation of real or generated user features and the real product embedding and decides whether the user feature is real or fake. This model is a fully connected network with two hidden layers and uses LeakyRelu as the activation function. The last layer uses a linear activation function to output the score indicating how real the users are based on the real product embedding. This information can be used to guide the generator to generate better fake users via backpropagation. If this value is small, the generated users are close to the real data.

To ensure that the model learns to generate users based on the product used for conditioning, we also forced the model to learn about the product by adding a reconstruction loss (L^p). The generator not only generates the users, but also the embedding used for conditioning. The reconstruction loss is the cosine distance between the input product embedding and the product embedding at the output. For the generated users, the loss function is the WGAN-GP loss. Thus, the overall loss function for the generator is the weighted sum of reconstruction loss and user generator loss that is defined as:

$$\propto L^u + (1-\propto)L^p \tag{39}$$

Where $\propto$ is a hyperparameter that we need to tune. L^u is a generation loss or loss function of generated users. L^p is a reconstruction loss or loss of generated products.

For hyperparameter tuning, we used grid-search to obtain the optimal configurations or hyperparameters of the models which provide the highest performance score for this dataset.

- For the generator noise, we tried different vector sizes (36, 64, 96, 128) and found that 64-dimension noise gives the highest performance for this model.
- The generator configuration uses a structure 64 → 128 → 256 as the hidden units. The last layer outputs a vector of length 52+70 = 122 which is the number of user and product features.
- The discriminator has two hidden layers 128 → 64 and the last layer is linear with size 1 to measure how real the input users are.
- τ in ST Gumbel Estimator, we used 0.9 to be the initial value with annealing rate of 0.005. The final value is 0.35.
- The optimizers, we tried (Adam, SGD, RMSprop) and found that Adam with beta = 0.7 gave the best result.
- In this work, the best ratio of number of times the generator is trained to the discriminator is 1:5.

CHAPTER V

EXPERIMENTAL RESULT

We performed experiments to verify the effectiveness of our generation system. In this section we will talk about the experimental setups and the results of each experiment.

5.1 Experiment Setup

We construct the training and test set by randomly select 50 products from the total products to be treated as novel products. The training set contains all products that are not selected for testing, meaning the test are completely unseen by the model. We repeat the selection 10 times to construct model 10 independent training and test sets.

5.2 Evaluation Metric

Our goal is to have the model predict the distribution of web logs given unseen products. Thus, we cannot measure the performance of the web log individually. We have the model generate 10,000 web log per test products and then measure the statistically properties of the log generated with respect to the ground truth. Prior work used Relative Similarity Measure (RSM) which captures the characteristic of each attribute of the generated product relative to other products [13]. However, this measure does not capture higher order statistics. Moreover, some use cases might require precise knowledge of the distribution rather than relative difference. Thus, we also propose three other metrics that can be used to measure the quality of the generated logs. The four metrics can be summarized as follow:

5.2.1 Relative Similarity Measure (RSM)

This measure is used to measure the relative similarity between real and generated samples by comparing between two products. The concept of computing this measure is shown in Algorithm I.

Algorithm I RSM metric calculation

Require: the list of testing products (p), the number of testing products (n_p), the real users statistics (U^r), the generated users statistics (U^g)

```
 1:  $n = 0$
 2:  $n_{count} = 0$
 3:  for $f$ in referring feature do
 4:      select $u^r == f$ and $u^g == f$ from $U^r$ and $U^g$ respectively
 5:      for each product pair in (i, j) do
 6:          select the $s_i^r$ and $s_j^r$ from $u^r$
 7:          select the $s_i^g$ and $s_j^g$ from $u^g$
 8:          if $(s_i^r > s_i^r \; and \; s_i^g > s_i^g)$ or $(s_i^r < s_i^r \; and \; s_i^g < s_i^g)$ or $(s_i^r = s_i^r \; and \; s_i^g = s_i^g)$
 9:              $n_{count} = n_{count} + 1$
10:          end if
11:          $n = n + 1$
12:      end for
13:      $rsm = n_{count} \div n$
14:      return $rsm$
15:  end for
```

An example in Figure 25, the first low level feature of referring is Facebook. The portion of real users that were referred from Facebook is 30% ($s^r_{A(facebook)}$) and 20% ($s^r_{B(facebook)}$) for product A and product B respectively. The $s^r_{A(facebook)}$ is 10% higher than $s^r_{B(facebook)}$. In the same way, the product A's generated users were referred from Facebook 50% ($s^g_{A(facebook)}$) and is also higher than $s^g_{B(facebook)}$. Thus, we count this as a relative similarity between real users and generated users. Similarly, the portion of real users that were referred from Google is 20% for product A ($s^r_{A(google)}$) and 50% for product B ($s^r_{B(google)}$). The $s^r_{A(google)}$ is lower than the $s^r_{B(google)}$. In comparison with generated users, the $s^g_{A(google)}$ is also lower than the $s^g_{B(google)}$. The last one is directing visitors. The real direct user is 50% ($s^r_{A(direct)}$) and 30% ($s^r_{B(direct)}$) for product A and product B respectively. The $s^r_{A\,(direct)}$ is higher than $s^r_{B\,(direct)}$. This is the same direction as the comparison between $s^g_{A\,(direct)}$ and $s^g_{B\,(direct)}$. Thus, we count the generated users as high relative similarity with the real users on all three low-level referring page features.

5.2.2 Correlation Coefficient (CORR)

This metric measures the strength of the relationship between real and generated samples as the formula:

$$\rho_{p_r p_g} = \frac{cov(p_r, p_g)}{\sigma_{p_r}\sigma_{p_g}} \quad (40)$$

where $cov(p_r, p_g)$ is the covariance of p_r and p_g. σ_{p_r} is the standard deviation of p_r. σ_{p_g} is the standard deviation of p_g. The value of CORR is between -1 and 1 where -1 means negative correlation and 1 means positive correlation. An example in Figure 25, the real users were referred from Google 20%, Facebook 30% and direct 50%. While the generated users were from Google 10%, Facebook 50% and direct 40%. The correlation between [0.2, 0.3, 0.5] and [0.1, 0.5, 0.4] is 0.58. If the correlation coefficient >= 0.85 which is defined as a high correlation in statistics, we count that as a high correlation between real and generated users.

Algorithm II Correlation coefficient metric calculation

Require: the list of testing products (p), the number of testing products (n_p), the real users statistics (U^r), the generated users statistics (U^g)

```
1:  $n = 0$
2:  $n_{count} = 0$
3:  for $f$ in referring feature do
4:      select $s_i^r$ from $U_i^r$
5:      select $s_i^g$ from $U_i^g$
6:      if $CORR(s_i^r, s_i^g) \geq 0.85$ then
7:          $n_{count} = n_{count} + 1$
8:      end if
9:  end for
10: $corr = n_{count} \div n$
11: return $corr$
```

5.2.3 Wasserstein distance or Earth mover's distance (EMD)

This metric measures the distance between two probability distributions as the formula:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} E_{(x,y)\sim\gamma}[\| x - y \|] \tag{41}$$

Where P_r is the probability distribution of real users, P_g is the probability distribution of generated users. $\Pi(P_r, P_g)$ is the set of all distributions. We use this metric to calculate the minimum cost of transforming the generated user distribution into the real user distribution. As Figure 25, the distance between real user distribution and generated user distribution is 0.2. The advantage of EMD is that even when two distributions are not overlaps, this measure can provide the distance value between two distributions. While Kullback-Leibler divergence (D_{KL}) provides the infinity when two distributions are disjoint.

5.2.4 Root Mean Square Error (RMSE)

This metric is the average of difference between proportion of real and generated users of each feature as the formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_r - p_g)^2}{n}} \tag{42}$$

where p_r is the proportion of real users, p_g is the proportion of generated users and n is the number of testing projects. RMSE measures how accurately the generated users are in the same unit as the data. Thus, this measure is easy for interpretation. The lower score is better. As Figure 25, the RMSE between [0.5, 0.3, 0.2] and [0.4, 0.5, 0.1] is 0.141 or 14.1%. It means that the difference between real and generated users is 14.1%.

To summarize the evaluation metrics as Figure 25, this is an example based on referring page feature. RSM score measures the relative similarity of real and generated users by comparing between two products. This example shows that the generated users are counted as high relative similarity with the real data. While the correlation coefficient (CORR) is used to measure the correlation of real and generated users within the same product. The correlation of this example is 0.58, we do not count as high correlation with the real data. The next one is Earth-

Mover distance (EMD) that measure the minimum cost to transform one distribution into the other. This shows that the cost of converting the generated distribution to the real distribution is 0.2. The last one is RMSE that shows the precision of the generator. These metric measures how close the proportion of generated users on each characteristic is to the real users.

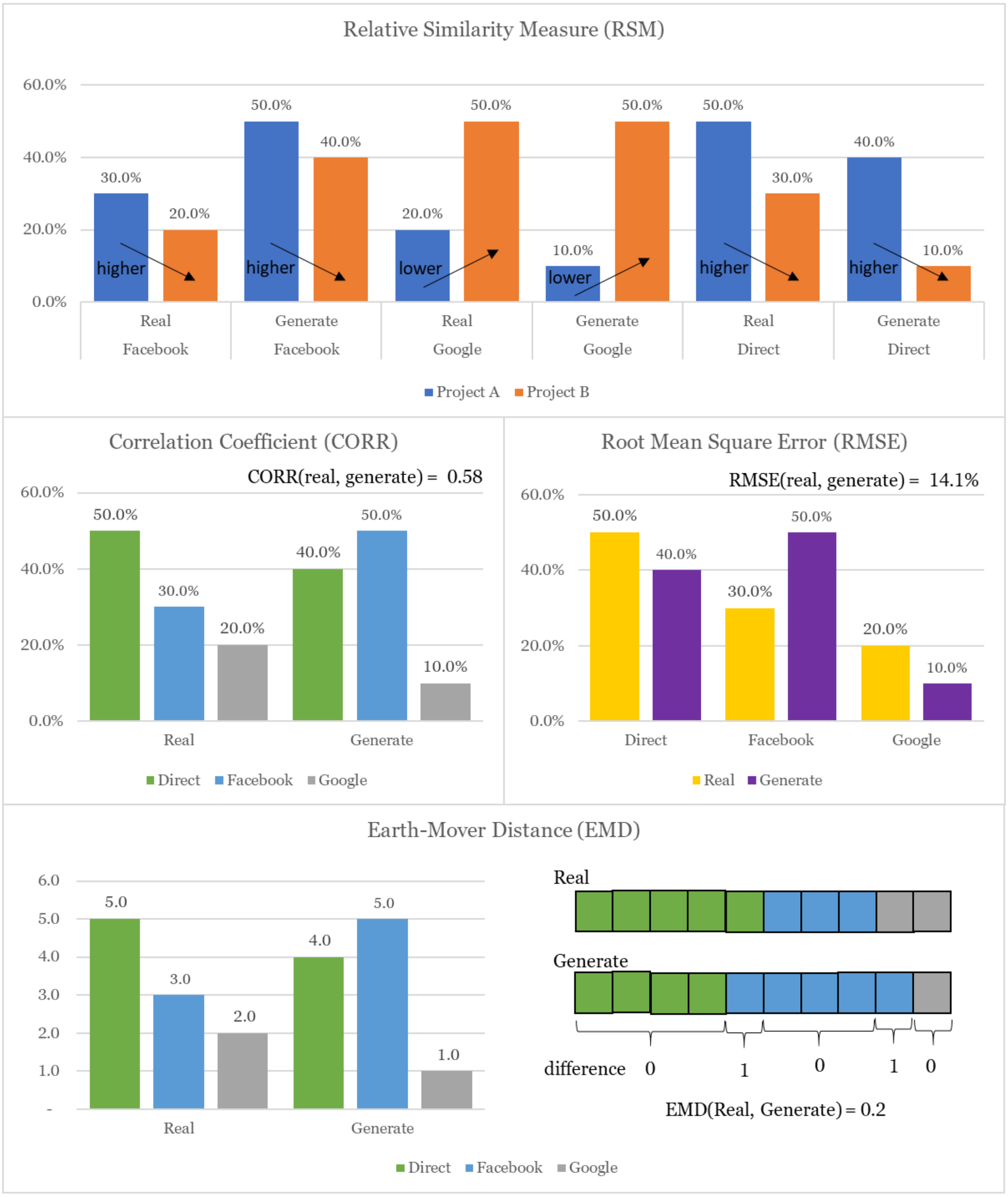

Figure 24 Example of our metrics

The metrics are measured on each feature and average. However, we can also measure the metrics in a multivariate manner (2 features). For example, RSM will be measure the relative similarity between a tuple of two features instead such as (referring page, weekday), (device, operation system), etc.

5.3 Baseline Product Embedding Model

We compare two kinds of model for learning our embeddings: ULMFiT with product description and Autoencoder with product features.

5.3.1 ULMFiT with product description

We used Universal Language Model Fine-tuning (ULMFiT) which was proposed in [38] to embed the product description. ULMFiT is a transfer learning for natural language processing (NLP) task such as text feature extraction and text classification. In Thai language, this model was pretrained with 60,005 embeddings by [39] which is part of pyThaiNLP. This model consists of 3 stages as the Figure 22.

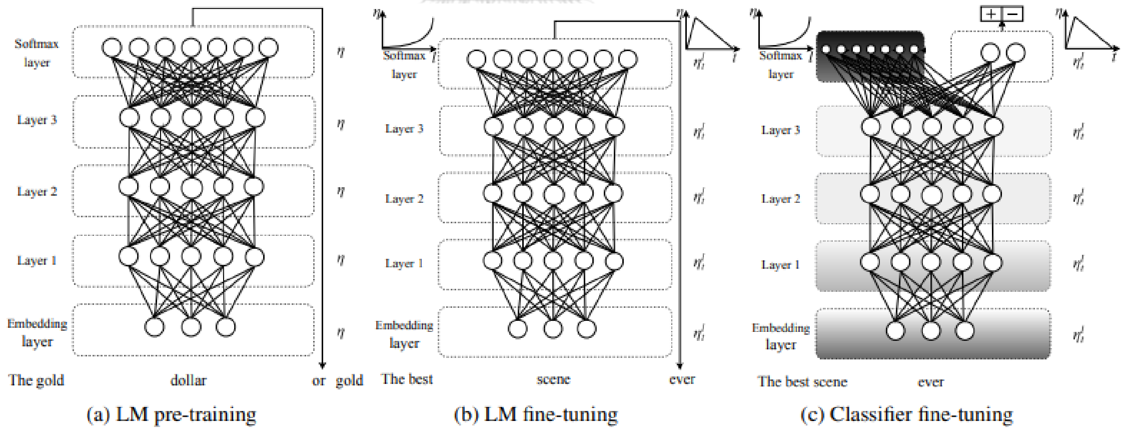

Figure 25 ULMFiT Model (Reference from figure 1 [38]*)*

- Language Model pre-training: this stage is training the language model on a general-domain corpus that captures high-level natural language features.
- Language Model fine-tuning: this stage is fine-tuning the pre-trained language model to learn task-specific features.

- Classifier fine-tuning: this stage is fine-tuning the classifier on the target task.

Instead of starting from random weights, we use the pre-trained language model which was provided in [40] and then fine-tune the language model on our product description. The last stage is a classifier. We use label from topic modeling which was trained by using Latent Dirichlet Allocation (LDA).

5.3.2 Autoencoder (AE)

The objective of the AE is to compress the original input and learn the best embedding that can be used to reconstruct the original input. The embedding is usually of lower dimension than the original input features so that the mapping is not trivial. We use a six-layer autoencoder with (256, 128, 96, 96, 128, 256) neurons at each layer, resulting in an embedding of size 64.

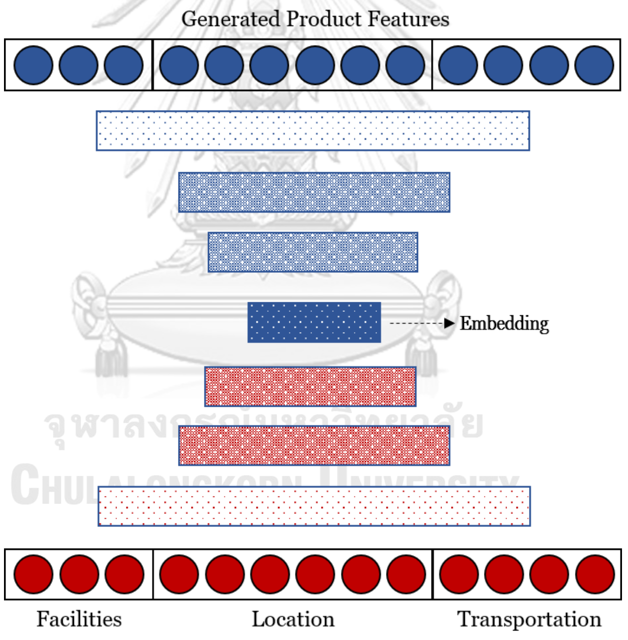

Figure 26 The product embedding from Autoencoder

5.4 Baseline Generation Approach

We compare our approach with two baseline approaches: nearest neighborhood approach and Conditional Variational Autoencoder (CVAE).

5.4.1 Nearest Neighbor (NN) approach

We apply the nearest neighbor concept to summarize the characteristics of users who are likely to be interested in new product. We select the user log of top 5 existing products that their characteristics are similar to the characteristics of new product. Thus, we know the list of possible values of each user feature for new product and sample based on that distribution.

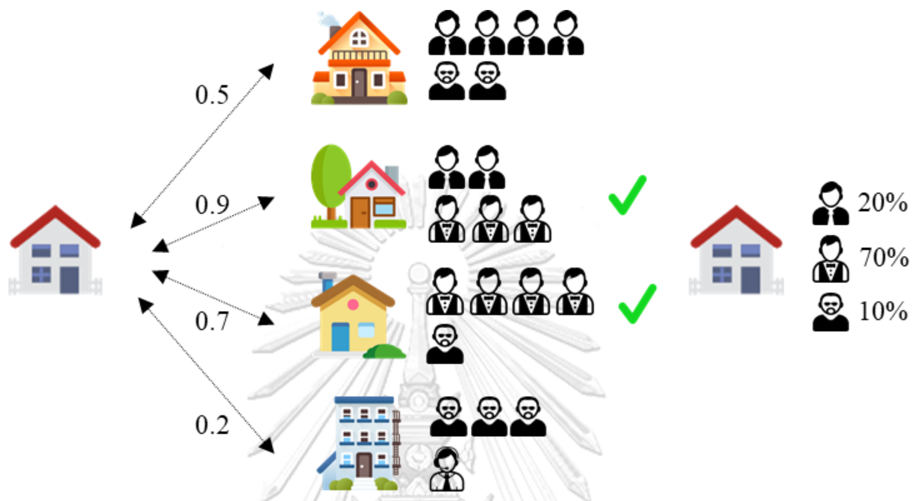

Figure 27 A nearest neighbor approach

5.4.2 Conditional Variational Autoencoder (CVAE)

CVAE is a deep generative model which is an extension of Variational Autoencoder [7]. This model was proposed in [8] and can control on the data generation process to generate some specific output by adding the additional information to both encoder and decoder. We use this model and add the product embedding to generate the user logs of new product.

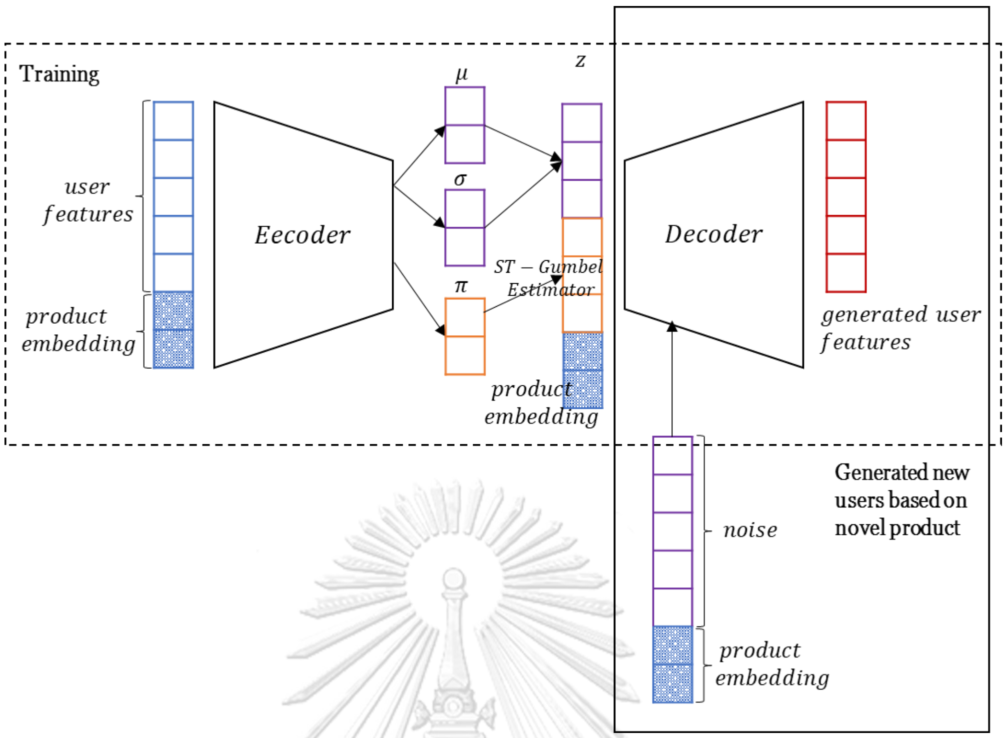

Figure 28 Conditional Variational Autoencoder approach

5.5 Results and discussion

The overall performance of our methodology (C-WGAN-GP) that is compared against several baseline approaches as shown in Table III and IV for 1 feature and 2 features respectively. Our proposed approach which used conditional GAN with embedding learned from recommendation system performed the best on every metrics. This shows the effectiveness of our approach in learning the distribution of novel products. The effectiveness of the learned embedding is shown when we compare different embeddings. For embedding from autoencoder, the original product features only improve the performance slightly on several metrics, but using recommendation embedding shows significant gain on all metric. While using the product description embedding from ULMFit is the worst performance because the product description is less information for capturing the relationship between products.

Table 3 The overall performance for 1 feature based on 4 metrics

Model	**1 Feature**			
	RSM	**CORR**	**EMD**	**RMSE**
C-WGAN-GP with REC Embedding	**72.5%**	**88.9%**	**0.59**	**16.2%**
C-WGAN-GP with AE Embedding	69.7%	87.8%	0.77	18.1%
C-WGAN-GP with ULMFiT Embedding	32.5%	78.7%	1.76	28.2%
C-WGAN-GP with Product Features	67.9%	86.6%	0.83	18.2%
C-VAE with REC Embedding	65.3%	85.6%	1.23	20.3%
NN with REC Embedding	54.7%	71.6%	1.69	28.0%

Table 4 The overall performance for 2 features based on 4 metrics

Model	**2 Feature**			
	RSM	**CORR**	**EMD**	**RMSE**
C-WGAN-GP with REC Embedding	**81.9%**	**82.5%**	**1.17**	**19.4%**
C-WGAN-GP with AE Embedding	78.8%	80.1%	1.98	21.4%
C-WGAN-GP with ULMFiT Embedding	64.5%	31.6%	6.95	15.8%
C-WGAN-GP with Product Features	78.4%	78.4%	2.56	21.6%
C-VAE with REC Embedding	72.2%	76.5%	3.27	23.8%
NN with REC Embedding	62.9%	39.8%	5.81	34.3%

Our simplest baseline is a nearest neighbor model (NN). The nearest products in the training set are used as the statistics of the novel product. This model uses cosine distance on the recommendation embedding to select the top 5 nearest products. Unsurprisingly, this method performs the worst, since for real estate there are rarely two products that are similar to each other.

We also trained another baseline based on CVAE with recommendation embedding. The CVAE baseline performed worse that all other GAN models, showing the effectiveness of GANs in learning the distribution of the customers.

As you can see in the Table IV, the performance of crossing two features is less than one feature because the model needs to capture the dependency between features.

We also show the performance of models for each feature as shown in Table V. The most interest is in the customer segmentation which has 36 possibilities and are high imbalance. The proposed model can get 75% CORR meaning that it can be used to give some guidance on what kind of customer would prefer the product. On the other hands, a NN approach which is something a human might have done based on his limited experience, would yield abysmal results.

Table 5 The performance of our approach for each feature

Feature	Model	RSM	CORR	EMD	RMSE
Device	**C-WGAN-GP with REC Embedding**	**69.3%**	**84.4%**	**0.059**	**10.0%**
	C-WGAN-GP with AE Embedding	54.5%	84.3%	0.081	10.5%
	C-WGAN-GP with ULMFiT Embedding	51.6%	84.3%	0.081	28.2%
	C-WGAN-GP with Product Features	55.1%	83.8%	0.085	10.5%
	C-VAE with REC Embedding	52.4%	83.4%	0.087	10.7%
	NN with REC Embedding	51.6%	83.2%	0.102	20.0%
Operation System	**C-WGAN-GP with REC Embedding**	**68.9%**	**99.5%**	**0.261**	**12.6%**
	C-WGAN-GP with AE Embedding	58.8%	97.9%	0.321	14.1%
	C-WGAN-GP with ULMFiT Embedding	51.9%	96.3%	0/362	22.4%
	C-WGAN-GP with Product Features	58.5%	97.9%	0.338	14.2%
	C-VAE with REC Embedding	57.7%	97.8%	0.357	14.3%
	NN with REC Embedding	49.5%	92.7%	0.431	15.9%
Customer Segmentation	**C-WGAN-GP with REC Embedding**	**80.4%**	**75.1%**	**2.091**	**24.4%**
	C-WGAN-GP with AE Embedding	75.6%	66.2%	3.945	51.5%
	C-WGAN-GP with ULMFiT Embedding	23.6%	22.4%	9.739	36.1%
	C-WGAN-GP with Product Features	73.1%	63.4%	4.175	52.8%
	C-VAE with REC Embedding	71.4%	60.4%	4.492	53.1%

Feature	Model	RSM	CORR	EMD	RMSE
	NN with REC Embedding	57.2%	1.5%	8.195	85.8%
Referring	**C-WGAN-GP with REC Embedding**	**68.8%**	**80.2%**	**0.148**	**19.4%**
	C-WGAN-GP with AE Embedding	62.4%	76.4%	0.238	24.5%
	C-WGAN-GP with ULMFiT Embedding	45.7%	75%	0.249	37.4%
	C-WGAN-GP with Product Features	61.8%	73.9%	0.259	26.4%
	C-VAE with REC Embedding	59.6%	78.6%	0.271	26.6%
	NN with REC Embedding	50.1%	54.7%	0.341	36.1%
Weekday	**C-WGAN-GP with REC Embedding**	**69.6%**	**100%**	**0.038**	**6.5%**
	C-WGAN-GP with AE Embedding	53.6%	100%	0.047	7.0%
	C-WGAN-GP with ULMFiT Embedding	52.5%	100%	0.046	21.5%
	C-WGAN-GP with Product Features	51.9%	99.1%	0.049	7.0%
	C-VAE with REC Embedding	50.7%	98.5%	0.052	7.1%
	NN with REC Embedding	46.2%	99.8%	0.068	7.1%
Time Period	**C-WGAN-GP with REC Embedding**	**73.9%**	**97.4%**	**0.094**	**8.7%**
	C-WGAN-GP with AE Embedding	62.7%	93.1%	0.113	9.7%
	C-WGAN-GP with ULMFiT Embedding	50.2%	94.3%	0.119	21.7%
	C-WGAN-GP with Product Features	57.8%	92.6%	0.119	10.1%
	C-VAE with REC Embedding	51.8%	90.6%	0.125	10.4%
	NN with REC Embedding	50.1%	96.3%	0.185	9.0%

We show one example of the generated distribution in Figure 28. Note how our model yields an estimate for customer segment number 15 as 1.2%, which is very close to the actual distribution of 1.7%. In a highly imbalance case such as this one, it is very hard for models besides GANs to uncover the long tail of the distribution. This shows the effectiveness of our model.

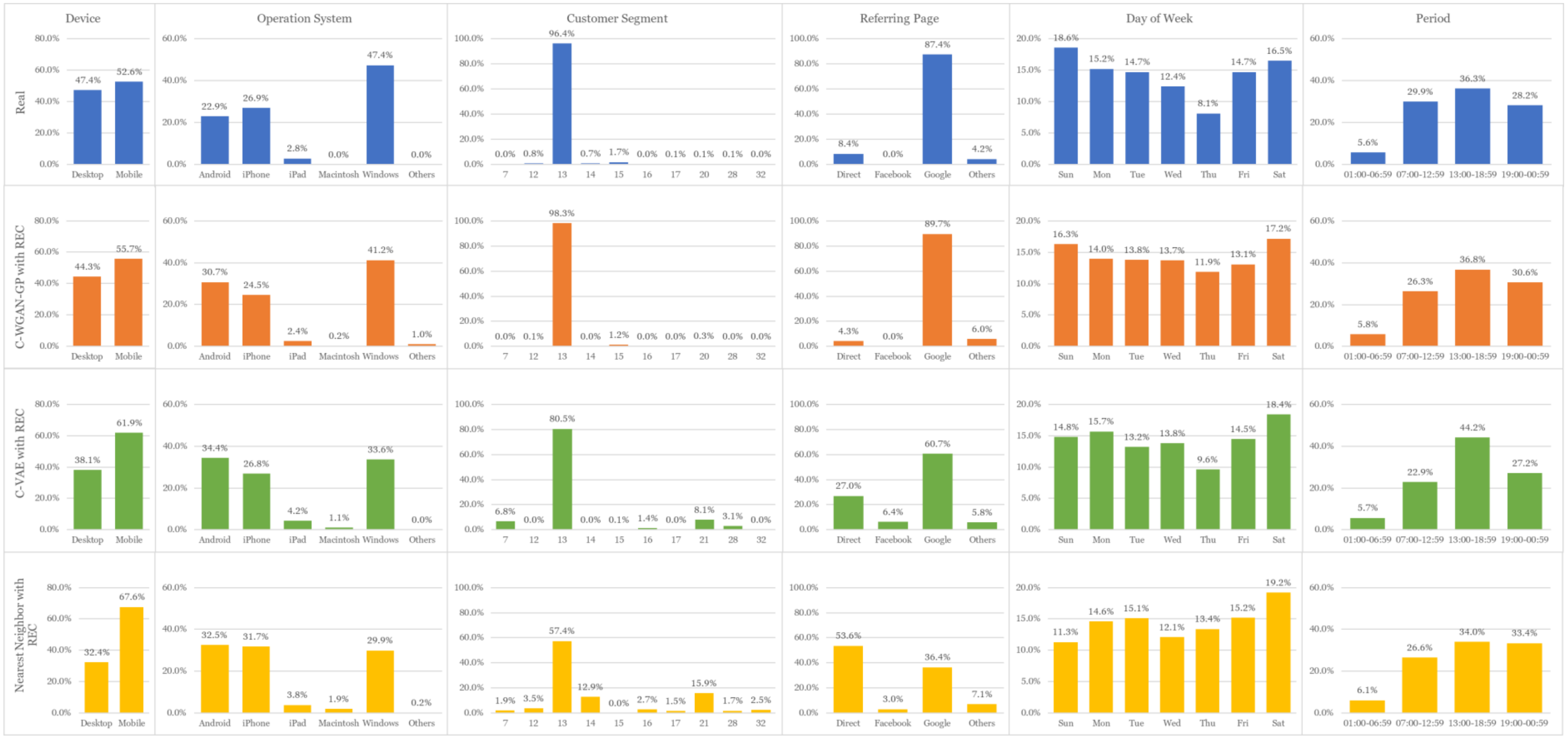

Figure 29 An example of the distribution comparison of each user features between the ground-truth users and generated users

CHAPTER VI

CONCLUSION

6.1 Conclusion

This work proposes a conditional generative adversarial network for generating realistic logs of user for novel product on the real estate domain. To improve the performance of model, one of key components is product embedding which we used to be an additional information to control the output of user generation. We propose to extract the product embedding via a trained recommendation system which is learned by using a Gated Recurrent Unit (GRU). The goal of embedding is to capture the relationship of similar product characteristic and user preference. Moreover, we handle the limitation of discrete outputs for GAN by using Straight-Through Gumbel Estimator which help simplify the pipeline and reduce error propagation of the models.

To evaluate the performance of our model, we compare our approach with two baseline approaches: nearest neighborhood approach and Conditional Variational Autoencoder (CVAE) by using four metrics to measure the quality of the generated logs: Relative Similarity Measure (RSM), Correlation Coefficient (CORR), Wasserstein distance or Earth mover's distance (EMD) and Root Mean Squared Error (RMSE). The results show that Our approach which used conditional GAN with embedding learned from recommendation system (C-WGAN-GP with REC Embedding) performed the best on every metrics. For the effectiveness of embedding, the result shows that the performance of embedding from autoencoder slightly improve when compare with the original product features, but using recommendation embedding shows significant gain on all metrics. For the effectiveness of generation approach, the results show that GANs is the best model to learn and capture the distribution of users although data is highly imbalance. Nearest neighbor model performed the worst because there are rarely two products that are similar to each other in real-estate domain. While CVAE performed worse than all other GAN models.

Our approach can generate users which are similar with the real users even highly imbalance case and performs better than the baselines.

6.2 Future work

- Demand Forecasting

  In this work, we did not cover the demand forecasting of new product before generating realistic users for knowing user characteristics. If we can forecast the demand, our approach will be completely useful for business planning when business launches new product.
- Product embedding

  In this work, the characteristics of product are separated into product features and product description. However, the embedding for product features and product descriptions are created from separated models. In future work it would be interesting to integrate these two embeddings into one for improving the performance.

REFERENCES

1. Suchacka, G. and G. Chodak, *Using Association Rules to Assess Purchase Probability in Online Stores*, in *Information Systems and e-Business Management*. 2017, Springer.
2. Iváncsy, R.a.V., Istvan, *Frequent Pattern Mining in Web Log Data.* Acta Polytechnica Hungarica, 2006. 3.
3. Hao, S., S. Zhaoxiang, and Z. Bingbing, *A User Clustering Algorithm on Web Usage Mining*, in *Electronics Instrumentation and Information Systems (EIIS)*. 2017.
4. Greenstein-Messica, A., L. Rokach, and M. Friedman, *Session-Based Recommendations Using Item Embedding*, in *Proceedings of the 22nd International Conference on Intelligent User Interfaces*. 2017, ACM: Limassol, Cyprus. p. 629-633.
5. Goodfellow, I.J., et al., *Generative Adversarial Nets*, in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. 2014, MIT Press: Montreal, Canada. p. 2672-2680.
6. Salman H. Khan, M.H.a.N.B., *Adversarial Trainingof Variational Auto-encoders for High Fidelity Image Generation.* CoRR, 2018. abs/1804.10323.
7. Deepak Pathak, P.K., Jeff Donahue, Trevor Darrell and Alexei A. Efros, *Context Encoders: Feature Learning by Inpainting.* CoRR, 2016. abs/1604.07379.
8. Wan, C., et al., *Crossing Nets: Dual Generative Models with a Shared Latent Space for Hand Pose Estimation.* ArXiv, 2017. abs/1702.03431.
9. Yu, J., et al., *Generative Image Inpainting with Contextual Attention*. 2018.
10. Zhu, J.Y., et al., *Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks*, in *IEEE International Conference on Computer Vision*. 2017. p. 2242-2251.
11. Kusner, M.J. and J.M. Hernández-Lobato, *GANS for Sequences of Discrete Elements with the Gumbel-softmax Distribution.* ArXiv, 2016. abs/1611.04051.
12. Yu, L., et al., *SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.* ArXiv, 2016. abs/1609.05473.
13. Kumar, A., A. Biswas, and S. Sanyal *eCommerceGAN : A Generative Adversarial Network for E-commerce*. ArXiv, 2018. arXiv:1801.03244.
14. Jang, E., S. Gu, and B. Poole, *Categorical Reparameterization with Gumbel-Softmax*. 2016.

15. Chris J. Maddison, A.M.a.Y.W.T., *The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables.* CoRR, 2016. abs/1611.00712.

16. Sohn, K., X. Yan, and H. Lee, *Learning Structured Output Representation Using Deep Conditional Generative Models*, in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. 2015, MIT Press: Montreal, Canada. p. 3483-3491.

17. Weng, L., *From GAN to WGAN.* ArXiv, 2019. abs/1904.08994.

18. Goodfellow, I., *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2016.

19. Baldi, P., *Autoencoders, Unsupervised Learning and Deep Architectures*, in *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning workshop - Volume 27*. 2011, JMLR.org: Washington, USA. p. 37-50.

20. Kingma, D.P. and M. Welling, *Auto-Encoding Variational Bayes*. 2013.

21. Arjovsky, M., S. Chintala, and L. Bottou, *Wasserstein GAN*. 2017.

22. Gulrajani, I., et al., *Improved Training of Wasserstein GANs*, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017.

23. Gumbel, E.J., *Statistical Theory of Extreme Values and Some Practical Applications*. 1954.

24. Maddison, C.J., D. Tarlow, and T. Minka, *A * Sampling*, in *Advances in Neural Information Processing Systems 27*. 2014.

25. Mikolov, T., et al., *Efficient Estimation of Word Representations in Vector Space.* 2013. abs/1301.3781.

26. Landthaler, J., et al. *Extending Thesauri Using Word Embeddings and the Intersection Method*. in *ASAIL@ICAIL*. 2017.

27. Barkan, O. and N. Koenigstein, *ITEM2VEC: Neural Item Embedding for Collaborative Filtering.* IEEE 26th International Workshop on Machine Learning for Signal Processing, 2016: p. 1-6.

28. Okura, S., et al., *Embedding-based News Recommendation for Millions of Users*, in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2017, ACM: Halifax, NS, Canada. p. 1933-1942.

29. Serban, I.V., et al., *A Hierarchical Latent Variable Encoder-Decoder Model for Generating Dialogues*, in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. 2017,

AAAI Press: San Francisco, California, USA. p. 3295-3301.

30. Zhao, T., R. Zhao, and M. Eskenazi. *Learning Discourse-level Diversity for Neural Dialog Models using Conditional Variational Autoencoders*. in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017. Vancouver, Canada: Association for Computational Linguistics.

31. Semeniuta, S., A. Severyn, and E. Barth. *A Hybrid Convolutional Variational Autoencoder for Text Generation*. in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017. Copenhagen, Denmark: Association for Computational Linguistics.

32. Dupont, E. *Learning Disentangled Joint Continuous and Discrete Representations*. ArXiv, 2018. arXiv:1804.00104.

33. Vo, T.V. and H. Soh, *Generation Meets Recommendation: Proposing Novel Items for Groups of Users*, in *Proceedings of the 12th ACM Conference on Recommender Systems*. 2018, ACM: Vancouver, British Columbia, Canada. p. 145-153.

34. Mirza, M. and S. Osindero, *Conditional Generative Adversarial Nets*. 2014.

35. Chen, X., et al., *InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets*, in *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016.

36. Larsen, A.B.L., et al. *Autoencoding Beyond Pixels Using a Learned Similarity Metric*. in *ICML*. 2015.

37. Bao, J.-m., et al., *CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training*, in *IEEE International Conference on Computer Vision (ICCV)*. 2017.

38. Howard, J. and S. Ruder. *Universal Language Model Fine-tuning for Text Classification*. in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018.

39. Polpanumas, C. *thai2fit (formerly thai2vec)*. 2017; Available from: https://github.com/cstorm125/thai2fit.

VITA

NAME	ปาริฉัตร ชลวิหารพันธ์
DATE OF BIRTH	30 April 1992
PLACE OF BIRTH	Bangkok
INSTITUTIONS ATTENDED	Faculty of commerce and accountancy, Chulalongkorn University
HOME ADDRESS	40/173 Pruksa Town Village Soi. Phetkasem 81, Nhong Kheam District, Bangkok 10160